NASA CR 170615

RESULTS OF DATA BASE MANAGEMENT SYSTEM PARAMETERIZED PERFORMANCE TESTING RELATED TO GSFC SCIENTIFIC APPLICATIONS

JUNE 30, 1983

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

Carol H. Carchedi, Thomas L. Gough, Herbert A. Huston

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

BUSINESS AND TECHNOLOGICAL SYSTEMS, INC.

BTS2-82-45/rd
1021

30 June 1983

RESULTS OF DATA BASE MANAGEMENT SYSTEM PARAMETERIZED PERFORMANCE TESTING RELATED TO GSFC SCIENTIFIC APPLICATIONS

Carol H. Carchedi
Thomas L. Gough
Herbert A. Huston

Prepared under
Contract No. NAS 5-26728

for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland 20771

by

BUSINESS AND TECHNOLOGICAL SYSTEMS, INC.
Aerospace Building, Suite 440
10210 Greenbelt Road
Seabrook, Maryland 20706

FOREWORD

This document summarizes the results of a variety of tests designed to demonstrate and evaluate the performance of several commercially available Data Base Management System (DBMS) products compatible with the Digital Equipment Corp. VAX 11/780 computer system. The tests were performed on the INGRES, ORACLE, and SEED DBMS products employing applications that were similar to scientific applications under development by NASA. The objectives of this testing included determining the strengths and weaknesses of the candidate systems, performance trade-offs of various design alternatives, and the impact of some installation and environmental (computer related) influences.

The study has been conducted under the technical direction of Regina V. Sylto of NASA's Information Extraction Division (IED) and her predecessor Elizabeth A. Martin. The document has been prepared by Thomas L. Gough (Project Manager), Carol H. Carchedi, and Herbert A. Huston of Business and Technological Systems, Inc. The authors wish to express their appreciation to Ms. Sylto and James Patrick Gary, also of the IED, for assistance in the final preparation of the document. In addition, the authors wish to thank Regina L. Barlow, of BTS, for her outstanding job of typing, editing, and preparing the document.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

Page

TABLE OF CONTENTS (cont'd)

1.0 INTRODUCTION

As part of a non-operational technology demonstration program, NASA's Office of Aeronautics and Space Technology (OAST) is conducting a Data Systems Technology Program (DSTA), formerly known as NASA End-to-End Data System (NEEDS), to demonstrate more efficient and timely transfer of data from the sensor to the user, for extraction of information by the user, and for exchange of information among users. Complementing the DSTA effort, NASA's Office of Space Science and Application (OSSA) is studying its data management workload with respect to a broad climate research program and has identified, among other things, data management requirements to support investigators within that program. The Information Extraction Division (IED) of Goddard Space Flight Center has numerous responsibilities associated with these two efforts, including the determination of a viable approach to maintaining and providing access to the various needed data bases. The IED has acquired several candidate Data Base Management System (DBMS) software packages for possible use in these efforts.

In order to evaluate these and future DBMS systems, this document defines a variety of tests which have been implemented to provide objective measures for evaluation of various aspects of DBMS performance. The purpose of this document is to report the results of these tests, not to summarize or draw conclusions from the tests. Also, the document was written to aid NASA in evaluating the performance of various data base designs applied to a number of NASA/GSFC scientific applications. Therefore, the DBMS performance testing specifications have been slanted to aid in the DBMS selection by using relevant NASA applications as the testbeds for the conduction of the tests.

The implementation of these test plans has initially been done to assist in the evaluation of three commercially available DBMSs, ORACLE, SEED, and INGRES, which operate on the DEC VAX 11/780 computer. Because this document is not intended as a tutorial, familiarity with and understanding of data base models, particularly the network and relational models, is necessary. Prior experience in using, or knowledge of, any of the three commercial products used in this testing is also desirable as an aid to increased understanding of the tests performed and the results documented.

1.1 Scope of Tests

The tests are directed at a variety of features that are frequently present in a general purpose data base management system. DBMS software has been developed in varying forms from various data base models for numerous applications. Implementations are based on hierarchical data models, network models, relational models, and other hybrid designs. Each approach may accomplish its goals in a different manner, but almost all are designed to collect information, present it to people or programs, and to update and delete it. A specification of tests, which are intended to measure DBMS performance in a manner which supports the comparison of one system to another, must not attempt to define test details which are too system specific. For example, if one were to define a test standard for measuring automobile performance, one would not wish to define a test which specifies the operation of a clutch pedal, because cars with automatic transmissions could not be included in such a test. Rather, one might wish to define such a test only to the point which describes the engagement of a gear which gives power to the drive wheels. Similarly, the wide variety of DBMS implementations requires that a test specification be general enough that it does not preclude its application to many DBMSs. An exception to this policy may occur when unique or radically variant features are present in a particular DBMS which must be addressed on an individual basis.

The specifications that are provided in Sections 2 and 3 deal with two different levels of testing. The first level deals with variations in data base structure and design, while the second level is devoted both to the impact of other data base and CPU users and to the tuning of data base and system parameters.

Two NASA applications, the Packet Management System (PMS) and the Pilot Climate Data Base Management System (PCDBMS), are under development and require the use of a DBMS to support their data management needs. Because of the scientific nature of the targeted data base applications, certain DBMS capabilities have been emphasized over others. For example, a typical NASA DBMS application might be one containing large volumes of "static" data, as in the PMS application. That is, the data base may contain scientific data which, once entered into the data base, requires very few changes. In this application, load and access performance is important, with emphasis on load performance. On the other hand, in an application where the data to be stored in the data base is "meta-informational" in nature (i.e. the data base consists of information about information, such as a catalog or directory), the information in the data base is subject to frequent change. The PCDB system is such an application. Here, update and delete performance must be assessed in addition to access rates, which are of utmost importance.

A large number of factors exist that influence DBMS performance. To identify and test all of these factors in combination and to determine their inter-relationships was considered beyond the scope of this plan for both schedule and budgetary reasons. Instead, these test specifications attempted to isolate a particular factor to determine its impact alone. The factor of data base size was de-emphasized in this specification because of prior work done in the evaluation of DBMS performance for the Information Extraction Division at GSFC. This work is described in a document entitled, "Data Base Management System Analysis and Performance Testing with Respect to NASA Requirements." In that report, a methodology is detailed for data base performance evaluation, which concentrates on general DBMS capabilities as a function of data base size.

These tests are quantitative in nature and are not designed to identify qualitative aspects of the systems under study. The results of these tests should not only provide a basis for system selection in the IED applications but should also provide a basis for predicting the impact of various data base designs under consideration using a single DBMS.

1.2 Test Environment

1.2.1 VAX Computer System Configuration

The test results summarized in this document were all generated from one of two DEC VAX 11/780 computer systems which were located at the Goddard Space Flight Center. The experimenters were somewhat at the mercy of the facility managers and an evolving hardware environment in this regard.

Initially, testing was begun on the MPP (Massively Parallel Processor) VAX 11/780 computer system. It was configured with three megabytes of memory and a variety of peripherals. Because the testing done on the MPP was done entirely in a controlled standalone mode, the relevant factor in the computer configuration was that the disk devices used to maintain the data base files and test software were RP06 units connected to the CPU by a single mass bus adapter. The operating system used during this testing was the VAX/VMS Version 2.2.

Tests were later shifted to the PC (Pilot Climate) VAX 11/780 computer system where approximately eighty percent of the tests described in this document were performed. This computer system began with the VAX/VMS Version 2.2, was upgraded to VMS Version 2.5, and was eventually upgraded to VMS Version 3.2. As these upgrades were installed, a few selected benchmark tests were repeated and no significant variations in performance were noted in the test results. The disk drive configuration was also upgraded during the course of the benchmark testing, beginning with two RP06 devices with the later addition of an RP07 512 megabyte disk device. From limited test iteration, no significant biases were noted due to changes in the disk configuration, primarily because the tests were performed in a standalone mode and the drives had similar seek and access rates. The PC VAX system was configured originally with two megabytes of main memory. About six months into the testing on the PC VAX, the memory was increased to four megabytes.

In summary, the test environment has been subject to alteration beyond the control of the experimenters but the biases of these variations have been monitored and are not felt to have contributed significantly to the measured results, thus minimally impacting the validity of any conclusions which might be drawn from the study.

1.2.2 Standalone vs. Contention Testing

All testing reported in this document was conducted in either a standalone mode or in a controlled contention mode. In most testing, it is necessary to control the environment so that valid comparisons can be made. The only way to ensure this system control is to obtain sole use of the computer system. The exception to this is in the testing for impact of contention on DBMS performance. Even in the contention testing, the runs were made in a standalone environment so that the only contention present was that introduced for the purpose of testing by those conducting the test. In other words, the contention was controlled. Unless specifically stated otherwise in this document, all testing was performed in a standalone environment.

1.2.3 HLI vs. Interactive

Most available DBMSs possess an interactive data manipulation language (DML) and query language and a parallel set of capabilities that can be invoked through an interface between the DBMS and a high level language such as FORTRAN or COBOL. These latter interface capabilities are frequently referred to as the Host Language Interface (HLI). In an effort to measure the variation in performance between an interactive DML and an HLI program performing the same function, various tests were conducted. In each of the three cases (ORACLE, SEED, and INGRES), neither the interactive DML nor the HLI program appeared to outperform the other. For this reason, the tests reported in this document were conducted using the HLI only to reduce the time required to gain results. This approach was further warranted because of the de-emphasis on the direct use of the DBMS provided query facilities in the preliminary designs of the PMS and PCDBMS applications.

1.2.4 Influence of NASA Requirements on Test Sequencing

The primary goal of the PMS application is to load 7 packetized header records per second into a data base. When testing was begun on the PMS application, no formal data base design had been developed. Therefore, a "prototype" design was initially used for testing. At that time, ORACLE 2.3.1 (and later 2.3.2) and SEED B.11.9 had been acquired and therefore, testing was performed using these two DBMSs. Variations in data base design were introduced on the "prototype" in an attempt to reach the goal of loading 7 header records per second.

It was intended that the results of this initial testing would be used as input for formulation of the actual PMS data base design, and indeed, this was the case. These preliminary results were responsible for design changes which are documented in Appendix I of this report.

In the PCDB application, the query rates are the primary concern of the data base developers. Therefore, while load rates were still of interest, testing focused on querying and various DBMS specific options available in querying (e.g. predicate ordering, sorted output, query nesting).

Throughout the testing, each time a new DBMS was acquired (or a new version of an existing DBMS), the testing was performed using the newest release. Some tests were repeated when it was believed that a new release of an existing DBMS might show significant improvement over an older version. The time and budgetary restrictions, however, dictated that not all tests could be performed using all versions of all DBMSs.

1.2.5 Terminology and Concepts

This section is devoted to giving the reader a clearer understanding of some of the terms and concepts included in this report. The section is further divided into VAX/VMS terms, applications terms, and DBMS specific terms.

1.2.5.1 VAX/VMS Terms

A discussion of the statistics which are presented in this document is appropriate here. The term connect time refers to the actual wall clock time which has elapsed. The CPU (Central Processing Unit) time is that amount of time which has actually been devoted to the processing of a task. A direct I/O is "an I/O (input/output) operation in which the system locks the pages containing the associated buffer in memory for the duration of the I/O operation. The I/O operation takes place directly from the process buffer."* A buffered I/O, on the other hand, is "an I/O operation (e.g. terminal or mailbox I/O) in which an intermediate buffer from the system buffer pool is used instead of a process-specified buffer."* A page fault is "an exception (interruption of the normal flow of instructions) generated by a reference to a page which is not in the process' working set (set of pages in memory)."*

1.2.5.2 Application Program Terms

In the PMS (Packet Management System), a packet is a unit of data from a spacecraft sensor or from a user which is to be archived. Each packet is prefixed by a packet header which identifies and describes that packet. This information is also stored. The header contains information such as mission ID, sensor ID, and source data format. The reader is referred to Appendix I for more information. A PMS burst is a group of packets. When the PMS data is being received for input to the data base, it will be transmitted a group or burst at a time. For this testing, a burst was identified as 72 header packets.

1.2.5.3 DBMS Specific Terms

This section is further broken down by DBMS, but not by version within that DBMS.

* Taken from the "VAX/VMS Summary Description", DEC, August 1978.

ORACLE DBMS

When a batch job is submitted which employs the ORACLE DBMS, a series of detached processes are created. Detached processes are subprocesses created by ORACLE and owned by the ORACLE account. In ORACLE Version 2.3, a detached process performs all of the DBMS related tasks. The batch job, or host job, performs the non-DBMS related tasks and the calls to the ORACLE routines. In ORACLE 3.0, while detached processes are still created, the host job performs most of the work.

In some of the ORACLE 3.0 testing, reference is made to an "old" space definition and a "new" space definition. These refer to the "initial" and "redefined" space definitions, respectively, as discussed in detail in Section 3.2.1.1.2 of this report. Unless specifically stated otherwise, the ORACLE 3.0 testing was performed using the new, or redefined, space definition.

An ORACLE page is equal to two VAX/VMS pages. In other words, an ORACLE page is 1,024 bytes.

An ORACLE buffer, or cache buffer refers to an internal buffer area used by ORACLE to increase the likelihood that often used pages are available in memory, thus eliminating the need for a disk I/O operation.

When reference is made to records being deleted in ORACLE Version 3.0, the reader should note that ORACLE deletes records logically, but in the current configuration, does not physically delete the records.

SEED DBMS

The SEED data base designer has direct control over the number of pages in an area and size of the pages. An area is the physical subdivision of the data base. In other words, an area is a file. The number of pages and size of the pages are specified in the schema definition. The only restriction on the page is that it must be a multiple of the VAX block size of 512 bytes (or 256 words). A SEED buffer is defined as the size of the largest page.

INGRES DBMS

As with the ORACLE DBMS, the INGRES DBMS creates a detached process when a batch job is submitted. A detached process is a subprocess. However, in INGRES, the detached process is owned by the host process' account, not by the INGRES account.

1.2.6 Reporter Statistics vs. VAX Accounting File

Two methods were used in presenting the results contained in this document. A brief explanation of each is given here.

The first method was the use of software which called the VAX system service routine GETJPI, and then produced a listing containing total connect time, total CPU time, total number of direct I/O's, and total number of page faults. The measurements did not include the opening and closing of the data base, nor did they include any preliminary FORTRAN code necessary for the successful operation of the software. The measurements did include, however, any other calls to data base routines, data base activity, and any other data base related FORTRAN code (such as formation of the primary key in the PMS application). In general, this was the preferred method of reporting results because only the data base related activity was measured. However, only the host process' activity was included in the "reporter" measurements. Any detached process activity was not included. In ORACLE 2.3 for example, the detached processes perform all of the data base related activity, so the statistics reported in this file are not a true indication of the actual measurements. The total connect time is the exception. Because the wall clock time is measured regardless of whether the host or detached process is performing the work, it is a true measurement in all DBMSs.

Because of the problem with reporting anything besides total connect time when using DBMSs which create detached processes, it was necessary to report some of the measurements using the VAX system accounting file, which keeps a log of all of the processes running on the system. This includes the batch processes and detached processes created by them. For each process, the total connect time, in seconds (in a later VMS version, hundredths of seconds), is given, along with the total CPU time in hundredths of seconds, the total number of direct I/O operations, the total number of buffered I/O's, and the total number of page faults. In DBMSs where a detached process is created by a batch job, the connect time reported is the higher of the connect time for the host job and detached process, while the CPU time, direct I/O's and page faults are the sum of those for the host process and the detached process. The measurements are given for the entire life of the process, so that in addition tc the data base related activity mentioned in the discussion of the "reporter" file, the opening and closing of the data base along with non-data base related FORTRAN code is measured.

While this is not as desirable as the "reporter" statistics because more than just data base related activity is measured, where necessary these statistics have been presented. Unless specifically stated otherwise, the results presented in this document have been extracted from the "reporter" file.

1.3 Summary of Tests

The purpose of this section is to summarize, briefly, the results which are presented in detail in the remaining sections of this document. In keeping with the intent of the report, no attempt is made to draw conclusions from the results.

The summary is presented in table form. The first column identifies the test performed. The remaining columns contain a code which represents the significance of the variation in results of the test performed as applied to each of the DBMSs under which it was conducted. There is one column for each of the following DBMSs - ORACLE 2, ORACLE 3, SEED, and INGRES. A blank space in any column indicates that the test was not performed using that DBMS. For a more specific version number, the reader is referred to the section containing the actual results of the test.

As mentioned above, the code represents the level of significance of the test on the performance of the DBMS. This level of significance was obtained from the "% Degradation" over a base scenario. The actual "% Degradation" (or a similar statistic) appears in most of the tables throughout the document. This percent was calculated by subtracting the results of the base run and the test run, and then dividing by the base run. The appropriate sign identifies whether there was improvement or degradation in the test run over the base run.

Three levels of significance and a code "NC" appear in the table. The code "NC" stands for no correlation and indicates that for the particular test performed, no trend in performance appeared to exist. The three levels of significance are signified by "L", "M", and "H". When the % degradation (or variation) between results was less than 10%, the significance of the test on data base performance was considered low, or "L". If the percentage was between 10 and 20, the level was interpreted as "M", for medium. Finally, if the percentage was greater than 20, the test which was performed was determined to have had a very significant or high ("H") impact on the data base performance.

The reader should be very careful in interpreting and comparing the results presented in the tables and throughout the report. The data base management systems are quite different (because for example, the network and relational concepts are quite different), and the test that was performed for one DBMS may not have been exactly the same test which was performed for another DBMS. The reader is referred to the specific sections of the report, where data base designs are described and where actual tests are defined.

The tables on the following pages summarize the tests which were performed and are documented in the remaining sections of the report. The tests appear in the same order as they do in the report. The tables are divided by section, with the section headings appearing above each table. The "Test Description" column defines the test and indicates what kind of measurement is reported (i.e. Load, Query, etc.). This column also states the appropriate sub-section numbers within the document. The levels of significance appear in the columns following the description. The levels are defined below each table. In columns where a range appears (for example, L-M), the level of significance may vary according to the exact test. For instance, in the test for "Number of Fields" in the first table, ORACLE 2 showed an L-M level of significance in both the load and query. The actual results in Section 2.1.2 show that for 1, 2, and 3 fields, the level of significance was low, but that the DBMS performance was moderately affected by the introduction of 7 fields. An "*" appearing next to the level of significance indicates that proper selection of the parameter tested impacts the level of significance of that parameter on DBMS performance. For example, in ORACLE 3 clustering, the performance may be greatly improved by creating the proper cluster. However, the performance may be greatly degraded by selection of an unsuitable cluster.

As stated previously, the reader is urged to examine the actual results of the tests in addition to using these tables as a guide to performance.

1.4 Organization of Document

The document is divided into four sections. In addition to this Section 1 introduction, Section 2 describes the first level of testing and Section 3 describes the second level of testing. The format used in these two sections includes a general description of a test plan and purpose followed by a description of the specific schema and test results for any DBMSs which have been subjected to that particular test. This approach was chosen because of the ease with which new DBMSs (or new versions of existing systems) could be subjected to one or more of the tests and be added to this document. In addition, another section has been added to this report. Section 4, titled "Supplemental Testing" is further divided by test. In this section, any additional testing that was performed which did not fall under headings in either Section 2 or Section 3 is reported. This includes, but is not limited to, further variations on schema design and alternate methods of querying the data base.

LEVEL 1 TESTING
RECORD/FIELD ALTERNATIVES AND RECORD STRUCTURES

Test Description		ORACLE 2	ORACLE 3	SEED	INGRES
Field Size (10 vs. 20 vs. 31 Bytes)					
	Load	NC		L	
(Section 2.1.1)	Query	L-H		NC	
Number of Fields (1x, 2x, 3x, 7x)					
	Load	L-M	L-M	NC-L	
(Section 2.1.2)	Query	L-M	L-M	NC	
Field Type (I*2 vs. C*2)					
	Load	NC	L	L	
(Section 2.1.3.1)	Query	NC	L	NC	
Field Type (I*2, I*4, R, C*2, C*4, C*8, C*12)					(I*2 vs. I*4)
	Load	NC		NC	NC
(Section 2.1.3.2)	Query	NC		NC	
Field Size vs. Number of Fields (3-16 Byte vs. 6-8 Byte vs. 12-4 Byte)					
(Section 2.1.4)	Load	M-H		L	
Field Size vs. Number of Fields (1-64 Byte vs. 4-16 Byte vs. 8-8 Byte vs. 16-4 Byte)					
	Load		M-H		L
(Section 2.1.4)	Query		L		L-H
Complexity of Data Base Design					
	Load	L		M	
(Section 2.2)	Query	L		H	

NC-No Correlation L-Low (< 10% difference) M-Medium (10-20% difference) H-High (> 20% difference)

LEVEL 1 TESTING
DIRECT ACCESS ALTERNATIVES

Test Description		ORACLE 2	ORACLE 3	SEED		INGRES
				Record	Index	
Overhead of key usage (Field not indexed vs. indexed)						
	Load	H	H	H	M	L-M
	Update	H	H	H	H	H
(Section 2.3.1)	Delete	NC	M	H	H	H
Key Length (4 vs. 8 vs. 12 Bytes)						
	Load	NC		L	L	
(Section 2.3.2)	Query	NC-L		NC	NC	
Duplication of Integer Keyed Value (1x, 2x, 5x, 10x)						
	Load	NC-L	NC-L	L-M	NC	NC-L
	Update	NC-L	NC-L	NC	L-M	NC-L
(Section 2.3.3)	Delete	NC-L	NC	NC	L	NC-L
Partial Duplication of Character Keyed Value ('constant, variable' vs. 'variable, constant')						
	Load	NC-L		NC-L	NC-L	
	Update	NC-L		NC-L	NC	
(Section 2.3.3)	Delete	NC-L		M	M	

NC-No Correlation L-Low (< 10% difference) M-Medium (10-20% difference) H-High (> 20% difference)

LEVEL 2 TESTING

CONTENTION WITH OTHER USERS

Test Description	ORACLE 2	ORACLE 3	SEED	INGRES
Contention with Non-DBMS Users				
Load & FORTRAN Compile	L	H	H	COPY - M REPEAT APPEND - H
1 Query & FORTRAN Compile	H	H	H	H
5 Queries & FORTRAN Compile	H	H	H	H
(Section 3.1.1)				
Contention with DBMS Users in Same DB				
5 Users	H	H	H	H
10 Users	H	H	H	H
15 Users	H	H	H	H
(Section 3.1.2.1)				
Contention with DBMS Users in Same & Different DB				
Query	M-H		M-H	
(Section 3.1.2.2)				
Contention Between Query & Load				
Load	M-H			
Query	H			
(Section 3.1.3)				

NC-No Correlation L-Low (< 10% difference) M-Medium (10-20% difference) H-High (> 20% difference)

LEVEL 2 TESTING
DBMS SYSTEM PARAMETERS

ORACLE 3	SEED	INGRES
Buffering (50 vs 100 vs 200 Buffers) Load - H Query - NC-L	Buffering (50,50,50) vs. (100,100,100) vs. (50,50,39) Load - H*	Secondary Indices Query - NC-L
(Section 3.2.1.1.1)	(Section 3.2.1.2.3)	(Section 3.2.1.3.1)
Space Definition (Default vs. Redefined) Load - L	Journaling Load - H	Journaling Load - M
(Section 3.2.1.1.2)	(Section 3.2.1.2.1)	(Section 3.2.1.3.3)
Clustering Load - L-H* Query - L-H*	Hashing Algorithm Load - NC	HEAP vs. HASH vs. ISAM Query - H*
(Section 3.2.1.1.3)	(Section 3.2.1.2.2)	(Section 3.2.1.3.2)
ODL Load Load - NC-L		Loading Alternatives Load - H
(Section 3.2.1.1.4)		(Section 3.2.1.3.4)

NC-No Correlation L-Low (< 10% difference) M-Medium (10-20% difference) H-High (> 20% difference)
* - proper selection impacts level of significance

LEVEL 2 TESTING

COMPUTER OPERATING SYSTEM PARAMETERS

Test Description	ORACLE 2	ORACLE 3	SEED	INGRES
Variation in VAX/VMS Behavior Load (Section 3.2.2.1)	DB Not Reinit. - L DB Reinit. - M		L	
Working Set Size Load (Section 3.2.2.2)		NC-L	L	
Disk Allocation Load (Section 3.2.2.3)	H		L	

NC-No Correlation L-Low (< 10% difference) M-Medium (10-20% difference) H-High (> 20% difference)

SUPPLEMENTAL TESTING

Test Description		ORACLE 2	ORACLE 3	SEED		INGRES
				Record	Index	
Number of Keyed Fields (4 vs. 3 vs. 1)						
	Load	M-H		H	H*	
(Section 4.1)	Query	NC		NC	H*	
Enlarged Data Base (50→100K)						
	Load		NC			NC
	Create Indices		NC			NC
	Query		NC			NC
(Section 4.2)	Delete Indices		NC			NC
Predicate Reordering						
(Section 4.4)	Query	L				L
Sorted Output						
(Section 4.5)	Query	H				H

NC-No Correlation L-Low (< 10% difference) M-Medium (10-20% difference) H-High (> 20% difference)

* proper selection impacts level of significance

2.C LEVEL 1 DATA BASE PERFORMANCE TESTING

The Level 1 testing consists primarily of measuring the performance impact of alternative data base designs. The variety of alternatives in a DBMS data base design is dependent upon the flexibility offered by its schema specification. A DBMS offering more options for schema specification naturally has an increased number of alternative designs possible for the data base and vice versa. Also associated with the Level 1 test plan are the specification of tests to assess the impact on indexing overhead and performance for fields of differing lengths and with different degrees of duplicate values present. All Level 1 testing was performed in a standalone environment using the PMS application.

2.1 Record/Field Alternatives

Information to be managed by a DBMS normally consists of groups of logically related items. One occurrence of a group is often referred to as a data record, and each item is considered a field within the record. An example might be a company's employee record with fields containing name, social security number, address, and wage rate. A DBMS may or may not manage the records presented to it in a manner that maintains the external form, order or adjacency. Schema design for a given application may require that some fields in the external records (i.e. input data records) be broken into separate records* internally to facilitate direct access or to minimize redundancy. For this reason, the term record and field must be clearly defined when mentioned in terms of a DBMS. In the context of this document the terms record and field apply to the externally managed data. When internal records are different, they will be identified as such in future discussions. If the terms are used to refer to the internal items, it will be so stated.

* The term records may not always be used by a DBMS but is intended here to refer generically to a logical collection of fields that are defined in the data base schema to comprise a single entity such as a "tuple" or row in the relational model or member and owner in a CODASYL system.

The following tests are dedicated to identifying the impact of different size records and field specifications, in terms of the number of fields and the number of bytes of data present.

2.1.1 Field Size

To determine the effect of field size on performance, a set of tests were designed in which a given field was first loaded with values which were all ten byte character strings. Values from this field were later retrieved. The test was repeated with twenty byte character strings instead of ten and then with thirty-one byte character strings. The tests were performed using a PMS-like application (see Appendix I) in which the MESSAGE field was used as the test variable. A total of 5,000 PMS headers were inserted into the data base and a total of 250 MESSAGE values were retrieved for each test.

ORACLE Version 2.3.2 Results

The table structure used for these tests is summarized by the following:

HEADER Table

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SDF	MESSAGE	HEADER

*Indexed Field

The average connect time for loading headers is summarized in the following table:

ORACLE LOAD RESULTS

MESSAGE Field	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
10 Byte Character	3.169	---
20 Byte Character	3.115	+1.7
31 Byte Character	3.166	+ .09

These results show no significant impact in loading performance over the range of field sizes chosen for the test. The fact that the thirty-one byte character string was inserted, on an average, faster than the twenty byte character string is attributed to variations in the operating system,not to the data base software.

The query which was repeated 250 times for each of the three cases is defined, and access rates are summarized below. The total connect time and average response time were der. ed from the reporter file, while the total CPU time was obtained from the VAX account file, where the corresponding connect times for the three runs (from the account file) were 62., 62., and 72. seconds, respectively.

SELECT MESSAGE FROM HEADER WHERE PRIMARY_KEY = ________

ORACLE QUERY RESULTS

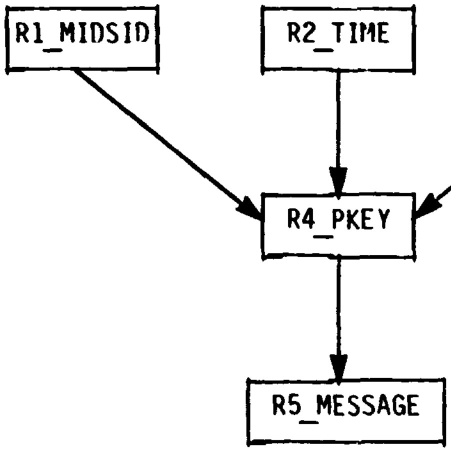
MESSAGE Field	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)
10 Byte Character	.15	---	37.72	38.14
20 Byte Character	.15	---	37.87	37.55
31 Byte Character	.20	+33.3	48.94	43.95

There does appear to be a significant degradation in response ir the third case. Test results indicated that an increase in CPU time was the cause of the degradation in the 31 byte character retrieval.

SEED Version C.00.02 Results

A diagram of the schema used for this testing is shown below.

SEED Schema



The R5_MESSAGE record consisted of one field, MESSAGE, which was defined as a 10 Byte Character, 20 Byte Character, and 31 Byte Character field for the three test runs, respectively.

Five thousand header records were first loaded into a PMS-like data base. The average number of headers loaded per second (connect time) for each of the three runs is shown in the table below.

SEED LOAD RESULTS

MESSAGE Field	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
10 Byte Character	5.03	---
20 Byte Character	5.00	+ .6
31 Byte Character	4.86	+3.4

While the trend of degradation appeared as the MESSAGE field size increased, the maximum degradation of 3.4% is not very significant. One explanation for this might be that the data base size remained constant for the three cases, while the record size of R5_MESSAGE varied, therefore causing more SEED page overflows.

The method used to access the message field was to do a sequential search (OBTNAP) on the first 5% of the R5_MESSAGE records. (See Appendix III) It appears, by looking at the results in the following table, that the queries are not significantly impacted by the length of the field.

SEED QUERY RESULTS

MESSAGE Field	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)
10 Byte Character	.007	---	1.84	1.51
20 Byte Character	.008	+14.3	1.89	1.52
31 Byte Character	.007	---	1.86	1.44

While there is a slight variance in the total connect time for the three runs, the mean response times are almost identical; that is, the vaiiations are insignificant and probably due to variations in the operating system software. While the "% Degradation in Average Response Time" for the 20 Byte Character run shows a 14.3% degradation, the degradation in total connect time is only 2.7%. The reason for the large degradation (14.3%) in average response time is that those averages have been rounded to three decimal places.

2.1.2 Number of Fields

To determine the effect of the number of fields present on the DBMS performance, a set of tests were designed in which the number of fields was changed each time. Each test loaded the data base and later retrieved a number of records from the data base. A PMS-like application (see Appendix I) was used for the tests. UIC field values were used as the repeated fields. Initially there was a single UIC field made up of two byte integer (I*2) variables. The test was repeated with a second UIC field, renamed of course, but with the same data values. Then it was repeated with a total of three UIC fields and finally with a total of seven fields. A total of 5,000 PMS headers were loaded each time and 250 queries were made of each data base.

ORACLE Version 2.3.2 Results

The table structures used for these tests are summarized by the following:

HEADER Table

A)

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SDF	MESSAGE	HEADER

HEADER Table

B)

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SDF	MESSAGE	HEADER	UIC2

HEADER Table

C)

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SDF	MESSAGE	HEADER	UIC2	UIC3

HEADER Table

D)

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SDF	MESSAGE	HEADER	UIC2	UIC3	UIC4	UIC5	UIC6	UIC7

*Indexed Field

The value in the UIC field is repeated in the added UIC"X" fields present. The query used thru the HLI to retrieve information retrieves all information in a data base row. The query used is:

```
SELECT * FROM HEADER WHERE PRIMARY_KEY = _____
```

The load results are summarized in the following table which shows the average insertion rate in headers per second for each of the tests:

ORACLE LOAD RESULTS

No. of UIC Fields	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
1	3.166	---
2	3.015	+ 4.7
3	2.929	+ 7.5
7	2.680	+ 15.4

The results indicate a consistent degradation in performance as more fields are added. Note that none of the fields added are indexed so the degradation is attributable to the addition of the field only. The percentage of degradation is noted in the table and shows how significant the impact is.

The results of the queries are summarized below for each of the data bases in the form of average response time:

ORACLE QUERY RESULTS

No. of UIC Fields	Average Response Time (Sec)	% Degradation in Average Response Time
1	.20	---
2	.20	---
3	.21	+ 5.0
7	.23	+ 15.0

It is apparent that these results also show a trend of degradation associated with the retrieval of the additional UIC fields. (The percent of degradation column is determined by using the single UIC field test as a baseline.)

Another set of tests described in the next section (Section 2.1.3.1) also illustrates the degradation in both loading and response when additional character fields are added in the same manner as the integer fields which were added in these tests.

ORACLE Version 3.0 Results

As in ORACLE 2.3, the header structure for the baseline run of one Integer*2 UIC field is shown below.

HEADER Table

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SDF	MESSAGE	HEADER

• Indexed Field

For the runs with 2, 3, and 7 UIC fields present, the table is identical to the one above except for the addition of the UIC fields. In all cases, the UIC field is non-indexed.

In running these tests, the space definition which was initially used was the old space definition described in Section 3.2.1.1.2. However, when a problem was encountered for the 7 UIC field run, the space definition was redefined to what is referred to as the new space definition. Because of the results in Section 3.2.1.1.2, the results presented here for the 7 UIC field run are probably "best case" results, as the redefined space definition was shown to have a small, positive effect on performance.

The results of loading the 5,000 record PMS-like data base appear below. The statistics are given in average number of headers inserted per second.

ORACLE LOAD RESULTS

No. of UIC Fields	Average Irsertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
1	5.04	---
2	4.76	+ 5.6
3	4.64	+ 7.9
7	4.28	+ 15.1

A steady degradation in performance appears as more fields are added to the header record. The same type of behavior appears to exist in querying the data base. Those results are shown below and are given in average response time in seconds. The query which was executed was:

SELECT * FROM HEADER WHERE PRIMARY_KEY =______

The query was performed 250 times, or for 5% of the records in the data base.

ORACLE QUERY RESULTS

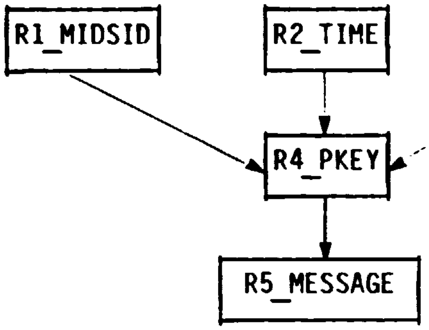
No. of UIC Fields	Average Response Time (Sec)	% Degradation in Average Response Time
1	.091	---
2	.093	+ 2.2
3	.095	+ 4.4
7	.101	+ 11.0

While the percent of degradation is not as great in querying as in loading, it still does appear, and the user should be aware that the addition of fields to a record may produce some degradation in both loading and querying.

SEED Version C.00.02 Results

The structure of the data base used to perform this testing is shown below:

SEED Schema



The UIC field(s) were present in the R4_PKEY record. Each field was defined as an Integer*2, named UIC1, UIC2, etc. None of the UIC fields were "CALC"ed.

The query that was used to retrieve the 250 R4_PKEY Records was to form the primary key, and "CALC" to the proper record (OBTNC) (See Appendix III).

The load results are shown in the table below.

SEED LOAD RESULTS

No. of UIC Fields	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate	Total Connect Time (Sec)	Total CPU Time (Sec)
1	4.74	---	1,054.93	538.27
2	5.04	-6.3	992.01	535.29
3	4.73	+ .2	1,057.16	537.52
7	4.92	-3.8	1,016.25	548.33

By looking at the column titled "Average Insertion Rate", it seems that there is no correlation between the number of fields present and the load rates. However, the "Total CPU Time" column shows, that while the CPU times for the 1 field, 2 fields, and 3 fields runs do not vary significantly, when seven fields are added, the load rate does suffer somewhat.

The results of querying on 250 R4_PKEY records are shown below.

SEED QUERY RESULTS

No. of UIC Fields	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)	% of Total Connect Time Devoted to CPU
1	.07	---	17.78	9.39	53
2	.09	+ 28.6	23.30	9.49	41
3	.07	---	17.35	9.31	54
7	.06	- 14.3	16.06	9.32	58

In this table, it appears that the average response time is better for the run with seven fields than the other runs. By looking at "Total CPU Time," there does not appear to be any degradation in retrieving a record through "CALC"ing to it, by having more fields present. After the main task of locating the beginning of the desired record is completed, the time to actually access the record is minimal and is not affected significantly by the length of the record. A direct I/O is issued for the record, independent of its length. In the run with two fields, the total CPU time is about 1-2% higher than the other CPU times. A table showing the total number of direct I/O's for each of the four runs follows. Looking at the number of direct I/O's issued shows that for the two field run there were 424 direct I/O's, where in the other three runs there were from 374 to 383 direct I/O's. This increase could account for the increase in total CPU time.

SEED QUERY RESULTS

No. of UIC Fields	Total No. of Direct I/O's
1	374
2	424
3	383
7	379

2.1.3 Field Type

DBMSs can manage data in a variety of forms including integer data as in the preceding section's UIC field, as real or floating point values, and as character data, to name the principal ones. To assess a variation in performance due to a field's data type, several tests were designed and conducted. In each test design, a PMS-like application (see Appendix I) was used.

2.1.3.1 Field Type - Test I

A set of tests similar to those in the preceding section were designed. The exception was to replace the integer data in the UIC fields with two byte character strings. The same loads and retrieves performed before were repeated so that both sets of test results could be compared to determine the difference, if any, in performance.

ORACLE Version 2.3.2 Results

As stated above, the same data base structure was used in these tests as was used in Section 2.1.2 with the exception that the various UIC fields were defined as character and not integer. The results of correspondingly similar tests (i.e. tests with equivalent numbers of UIC fields) should reveal impacts of the different data types.

The table below summarizes the average insertion rates for the corresponding tests:

ORACLE LOAD RESULTS

No. of UIC Fields	Average Insertion Rate (Hdrs/Sec)		% Improvement Integer over Character
	Character*2	Integer*2	
1	3.141	3.166	+ .8
2	3.077	3.015	-2.0
3	2.988	2.929	-2.0
7	2.708	2.680	-1.0

These results show that no statistically conclusive evidence exists to indicate that there is a significant difference between the two sets of tests.

The query results are summarized in a similar fashion below showing the average response times:

ORACLE QUERY RESULTS

No. of UIC Fields	Average Response Time (Sec)		% Improvement Integer over Character
	Character*2	Integer*2	
1	.18	.20	-11.1
2	.21	.20	+ 4.8
3	.20	.21	- 5.0
7	.23	.23	---

Like the load results, the query results demonstrate no evidence to support the conclusion that there exists a marked difference in performance due to data type alone.

ORACLE Version 3.0 Results

The data base design used in this testing was identical to that described in Section 2.1.2 except that the UIC fields were defined as 2 byte character fields instead of 2 byte integer fields. Also as in Section 2.1.2, the old space definition was used for the testing of 1, 2, and 3 fields while the new space definition was used for the 7 field run.

In the table below, the average number of PMS header records inserted per second is shown for character vs. integer data for each of the four tests conducted, along with the percent difference between the two types of data for each test.

ORACLE LOAD RESULTS

No. of UIC Fields	Average Insertion Rate (Hdrs/Sec)		% Improvement Integer over Character
	Character*2	Integer*2	
1	5.12	5.04	-1.6
2	4.97	4.76	-4.2
3	4.76	4.64	-2.5
7	4.54	4.28	-5.7

A slight improvement appeared in each case when the UIC field type was defined as character instead of integer type data.

The same type of table appears below comparing query results when UIC was defined as character versus integer.

ORACLE QUERY RESULTS

No. of UIC Fields	Average Response Time (Sec)		% Improvement Integer over Character
	Character*2	Integer*2	
1	.087	.091	- 4.6
2	.086	.093	- 8.1
3	.088	.095	- 8.0
7	.093	.101	- 8.6

As in the load, the average response time in each case was slightly better for the character fields as opposed to the integer fields.

In any ORACLE field which has been defined as NUMBER (i.e. when UIC is I*2), each time an insert operation is performed, the value to be inserted into the table is converted to ORACLE'S internal storage format. This format is variable length extended precision floating point format. Likewise, each time a select operation is performed, ORACLE must convert a NUMBER field from internal storage format to external format. On the other hand, ORACLE stores character type data in ASCII, so no conversion is necessary on inserts or selects.

If conversion can be considered an explanation for the differences noted above, one would expect to see the difference appear in CPU time as opposed to I/O operations or page faults. Below, two tables appear which show the total statistics for character versus integer data in both the load and query runs.

ORACLE LOAD RESULTS

UIC Description	No. of UIC Fields	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
Character	1	975.83	656.66	10,923	26,743
	2	1,005.35	685.01	11,044	27,357
	3	1,050.71	714.35	11,181	30,717
	7	1,101.00	770.60	10,910	20,936
Integer	1	992.39	661.17	10,847	25,510
	2	1,049.95	703.70	11,168	28,720
	3	1,078.38	737.47	11,165	28,815
	7	1,167.09	830.12	11,181	23,527

ORACLE QUERY RESULTS

UIC Description	No. of UIC Fields	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
Character	1	21.68	13.91	537	2,416
	2	21.48	13.77	524	2,606
	3	22.08	13.87	528	2,227
	7	23.13	14.23	537	2,262
Integer	1	22.87	14.85	525	2,754
	2	23.34	15.30	511	2,502
	3	23.64	15.46	517	2,718
	7	25.21	16.56	522	2,882

While, in both tables, the total number of direct I/O's and page faults vary and do not seem dependent on the UIC data type, the total CPU time is consistently higher when UIC was defined as integer than when it was defined as character.

While the actual differences in "Average Insertion Rate" and "Average Response Time" between character and integer fields are small, indicating very little true variance between the two data types, the pattern does exist and should be noted. While the evidence supporting the conversion theory is not conclusive, this may be a contributing factor in the differences which appeared.

SEED Version C.0C.02 RESULTS

The design of the data base used in this testing was identical to the design in the previous section with the exception that the UIC field(s) were designated as Character*2 instead of Integer*2. The same query was executed in this testing. Because the results in the previous section showed that there was no impact of number of fields on response time, a new query was designed and tested to see if it better measured the impact of additional fields. In this query, the primary key is formed, and then a sequential search (OBTNAP) is done on .5% of the R4_PKEY records. (See Appendix III) The results of both types of queries appear in the table showing query results.

The following table shows a comparison between the load rates for one to seven UIC fields of Character*2 and Integer*2 data type.

SEED LOAD RESULTS

No. of UIC Fields	Average Insertion Rate (Hdrs/Sec)		% Improvement Integer over Character
	Character*2	Integer*2	
1	4.83	4.77	- 1.2
2	5.11	5.07	- .8
3	5.11	4.77	- 6.7
7	5.14	4.97	- 3.3

The results in the table show that there is degradation in loading integer type data as opposed to character type data. However, the difference is slight and might be attributed to operating system software variations.

In the table below, the results of the previous query on Integer*2 type data is compared with the results of the two types of queries performed on the Character*2 data. Results are given in average response time.

SEED QUERY RESULTS

No. of UIC Fields	Average Response Time (Sec)			% Improvement Integer over Character
	Character*2, Seq. Search	Character*2, Calc PKEYs	Integer*2, Calc PKEYs	
1	17.3	.07	.07	---
2	17.7	.07	.09	- 28.6
3	17.0	.07	.07	---
7	17.2	.07	.06	+ 14.3

While the two figures in the "% Improvement" column indicate large differences in response time between character and integer type data for two and seven fields, respectively, these figures should be examined more closely. In the table below, the average CPU time (as opposed to connect time) is shown for the character data vs. the integer data for from one to seven UIC fields.

SEED QUERY RESULTS

No. of UIC Fields	Average CPU Time (Sec)		% Improvement Integer over Character
	Character*2, Calc PKEY's	Integer*2 Calc PKEY's	
1	.037	.038	- 2.7
2	.038	.038	---
3	.037	.037	---
7	.036	.037	- 2.8

The response time does not vary significantly between the character type data and the integer type data on the query of PKEY's.

While the sequential search query is much slower than the direct query on PKEY, there still does not seem to be any correlation between the number of UIC fields and the average response time. There is a maximum difference of about 4% in the average response times using the sequential search query.

2.1.3.2 Field Type-Test II

A second set of tests were designed which used only a single UIC field. This testing was conducted originally using ORACLE 2.3 and SEED. In each test the type of data loaded into the UIC field was varied. The first test used a UIC field that contained only integer values requiring less than 16 bits to represent them (equivalent to I*2 specification in FORTRAN). The second test used integer values greater than 16 bits in the UIC field (equivalent to I*4 specification in FORTRAN). The third test employed floating point (real) values in the UIC field. The fourth thru seventh tests used character data. In the fourth, a two byte character string was used; in the fifth, a four byte character string was used; in the sixth, an eight byte character string was used; and in the seventh test, a twelve byte character string was used. Each test consisted of loading 5,000 records and querying 250 of them. A PMS-like application was employed during these tests (see Appendix I).

When the INGRES Version 1.3 DBMS became available for testing, a similar but much less exhaustive test was performed to assess the impact of field type on performance. In this testing, a comparison was made between an I*2 field and an I*4 field only. A PMS-like data base was loaded with 5,000 records and no query was performed.

ORACLE Version 2.3.2 Results

The table structure used in each of the tests is summarized in the following diagram:

HEADER Table

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SDF	MESSAGE	HEADER

*Indexed Field

The table below shows the average insertion rate for loading 5,000 header rows into the table for each of the tests:

ORACLE LOAD RESULTS

UIC Data Description	Average Insertion Rate (Hdrs/Sec)	% Improvement over Slowest Load (I*4)
Integer (I*4)	3.086	---
Real	3.111	+ .8
Character (4 Byte)	3.133	+ 1.5
Character (2 Byte)	3.141	+ 1.8
Character (12 Byte)	3.158	+ 2.3
Integer (I*2)	3.166	+ 2.6
Character (8 Byte)	3.207	+ 3.9

The results in the table are summarized in order of slowest to fastest insertion rate. No obvious trend is detectable in the results. The difference between worst and best results (I*4 and 8 Byte Character data types) shows slightly less than four percent improvement. The results suggest that one should conclude the impact of data type on load performance is negligible.

Each of the tests included the querying of two hundred and fifty PMS headers employing the host language interface capability using the following query:

```
SELECT * FROM HEADER WHERE PRIMARY_KEY = __________
```

The results of these queries have been averaged for each test and are summarized below:

ORACLE QUERY RESULTS

UIC Data Description	Average Response Time (Sec)
Character (2 Byte)	.18
Character (12 Byte)	.18
Integer (I*4)	.19
Character (8 Byte)	.19
Integer (I*2)	.20
Real	.20
Character (4 Byte)	.20

The test results do not indicate a significant difference or noticeable trend in performance due to the specification of the data type for the range of types chosen in this test.

SEED Version C.00.02 Results

The data base design for this testing was identical to the design shown in Section 2.1.2, except that the data type of the UIC field was varied. The queries that were performed were the same two queries as in the previous section, one to form the primary key and "CALC" to the proper R4_PKEY (OBTNC) and the other to form the primary key and do a sequential search on the proper R4_PKEY (OBTNAP). (See Appendix III)

The average connect time for loading the various data types is shown in the following table:

SEED LOAD RESULTS

UIC Field Type	Average Insertion Rate (Hdrs/Sec)	% Improvement over Slowest Load (Real)
Real	4.74	---
Integer (I*2)	4.77	+ .6
Character (2 Byte)	4.83	+ 1.9
Character (8 Byte)	4.90	+ 3.4
Character (12 Byte)	4.90	+ 3.4
Integer (I*4)	4.97	+ 4.9
Character (4 Byte)	5.00	+ 5.5

The results show that there is no significant impact on loading rates due to the data type of a variable. There is a maximum difference of .26 insertions per second or about a 5% difference.

The query results for both queries are shown below, and it can be seen from the table that, as in loading, there is no significant impact on query rates due to data type variation. In the "CALC 5% PKEYs" query, while the average response time for the 12 Byte Character run appears to be greater than for the other runs, a look at the CPU times shows that for all seven runs, the average CPU time was .04 seconds. In the sequential search query, there is a negligible maximum difference in average query rates of about 3%.

SEED QUERY RESULTS

UIC Field Type	Average Response Time (Sec)	
	CALC 5% PKEYs	Seq.Search .5% PKEYs
Integer (I*2)	.07	17.7
Integer (I*4)	.07	17.5
Real	.07	17.4
Character (2 Byte)	.07	17.3
Character (4 Byte)	.07	17.7
Character (8 Byte)	.07	17.5
Character (12 Byte)	.1	17.8

INGRES Version 1.3 Results

The table structure used in this testing is shown here:

HEADER Table

MID_SID	SSC	SDF	TIME	UIC	MESSAGE	HEADER

In this test, a PMS-like data base was loaded two times. In the first run the UIC field was defined as Integer*2 and in the second run, the UIC field was defined as Integer*4. In each run, 5,000 records were loaded.

A table showing the insertion rate for the two runs is given below. The insertion rate includes the time used for loading and creating the necessary indices (MID_SID, TIME, and the concatenated primary key made up of MID_SID, SSC, SDF, and TIME were indexed).

INGRES LOAD RESULTS

UIC Data Description	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
Integer (I*2)	10.85	---
Integer (I*4)	10.89	- .4

There does not seem to be any correlation between load rate and field type in this test.

2.1.4 Field Size vs. Number of Fields

Sections 2.1.1 and 2.1.2 of this report dealt with the impact on performance of field size and number of fields, respectively. In the testing in each of those sections, as the number of fields grew or the size of the fields grew, the total size of the record grew. The purpose of the testing in this section was to assess the impact on performance due to the "trading off" of field size and number of fields. In other words, in each test, the total size of the records remained the same, while the number of fields and field sizes were varied.

The first test which was conducted employed the original PMS design as discussed in Appendix I of this report. In the appendix, reference is made to 384 bits (48 bytes) of information to be stored as a single field. In this test, those 48 bytes were broken down into multiple fields. For example, in one case 3-16 byte fields were created. This test was conducted using ORACLE and SEED.

The second test which was conducted employed the revised PMS design. This is also discussed in Appendix I. In this test, the length of and number of UIC fields were altered. This testing was performed using ORACLE and INGRES.

ORACLE Version 2.3.1 and ORACLE Version 3.0 Results

The first test in this section was performed using ORACLE 2.3.1 on the original PMS design which is shown below. For this test, there was interest in evaluating the impact of making the SECHDR field into several smaller fields. The SECHDR field does not have a rigidly defined format and was therefore treated as a single large field that could be retrieved by a user who was aware of the true subfield structure and who could decode the 384 bits accordingly. It was requested that tests be conducted to see what impact would be present if the data in the SECHDR was stored as multiple fields to permit access to smaller amounts of data at a single retrieve.

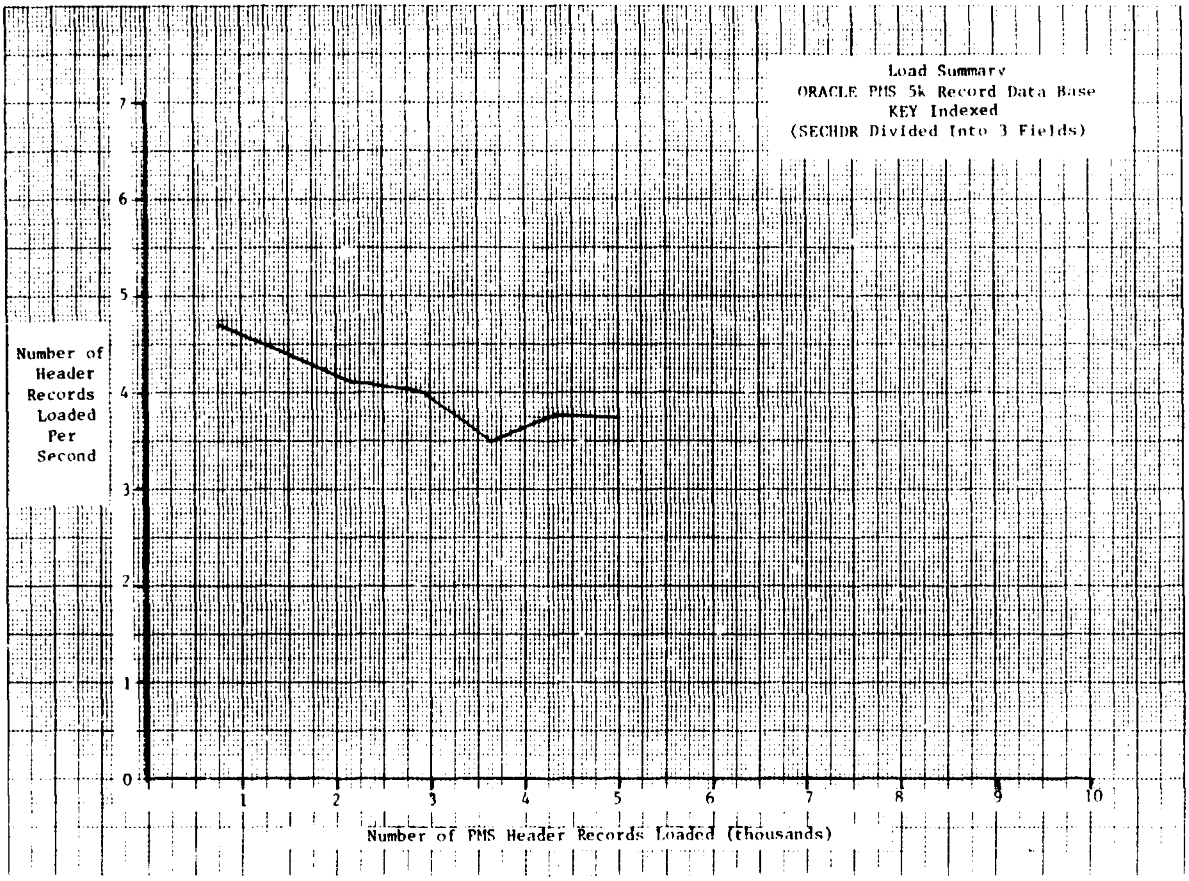
HEADER Table

KEY*	SID	MID	SSC	PLP	SDF	SHID	SIEC	TIME	PLS	SECHDR	UIC	COMMENT

* Indexed Field

The SECHDR was subdivided into three fields in one test, six fields in a second, and twelve fields in a third. In each of these tests the total lengths of the subdivided fields were equal (128 bits per field, 64 bits per field, and 32 bits per field respectively). The tests consisted of loading 5,000 PMS headers into an empty data base. None of the fields which constituted part of the SECHDR field were indexed. Only the KEY field was indexed for these tests. The results of the first test are plotted on the graph below.

The second and third of these tests (with 6 and 12 fields for SECHDR, respectively) are summarized in the graphs on the next page. The initial load performance in each of these tests continues to drop as more fields are added even though the total volume of data remains the same. The initial 720 headers were loaded in an average of 4.75 headers per second in the first test, at 3.8 in the second, and at 3.6 in the third. The degradation is less pronounced through the entire 5,000 record load but nevertheless as more fields are present there exists a reduction in the



average load rate per second. This could be anticipated since ORACLE treats each field present independently by adding a header to each describing the field number and length.

The results of the three runs are summarized in the table below, making the percent of degradation more apparent.

ORACLE LOAD RESULTS

Field Description	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
3-16 Byte Fields	4.01	---
6-8 Byte Fields	3.54	+ 11.7
12-4 Byte Fields	3.14	+ 21.7

The second test which was run to assess the field size versus number of fields was run under ORACLE 3.0. In this testing, the revised PMS design was applied, with the header table defined as:

HEADER Table

PRIMARY_KEY*	MID_SID	TIME	UIC	SDF	MESSAGE	HEADER

* Indexed Field

The UIC field was selected for testing. In all runs, the UIC field was a non-indexed field. First, UIC was defined as 1 field, 64 bytes in length. Next, UIC was defined as 4 fields, each 16 bytes in length. Following this, UIC was defined as 8 fields of 8 bytes each, and finally UIC was defined as 16 fields of 4 bytes each.

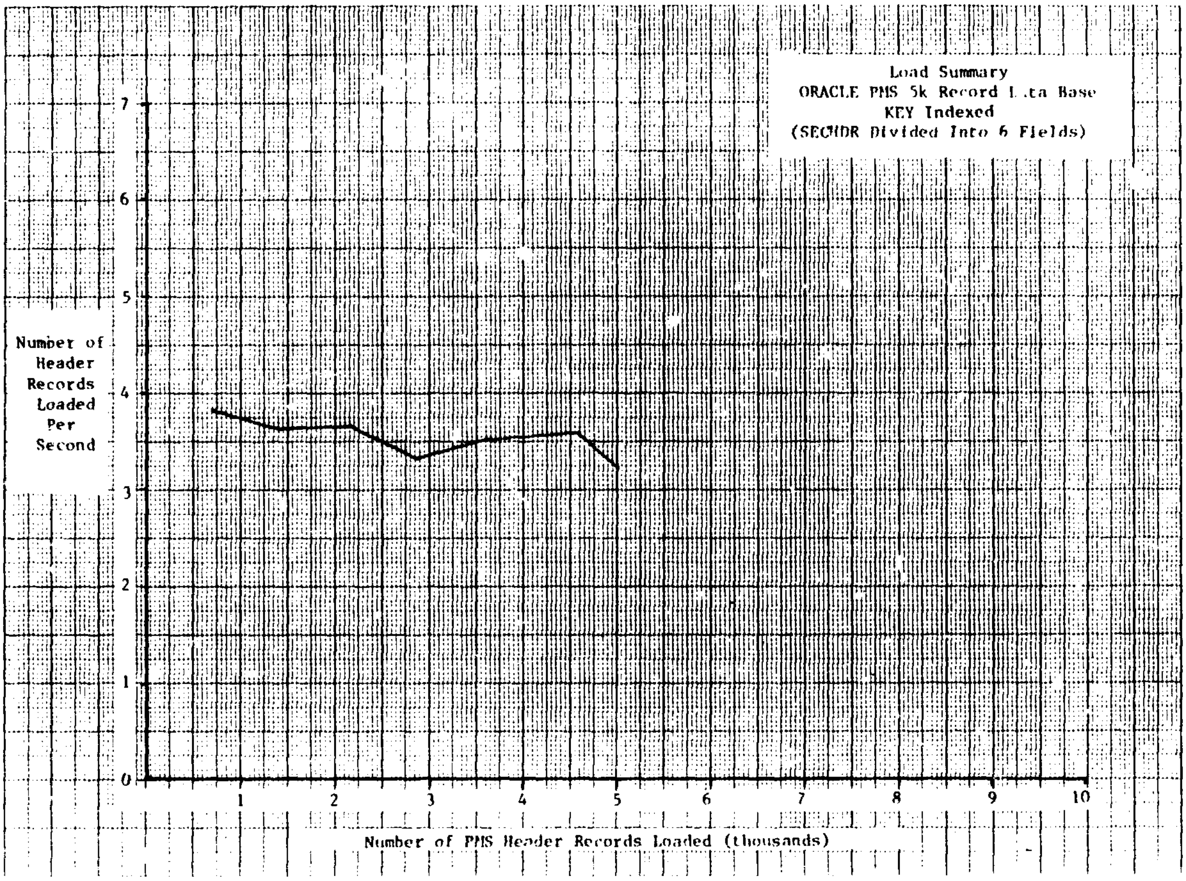
First, each of the four data bases was loaded with 5,000 header records. Following the load, the query listed below was performed on 5% or 250 records in the data base.

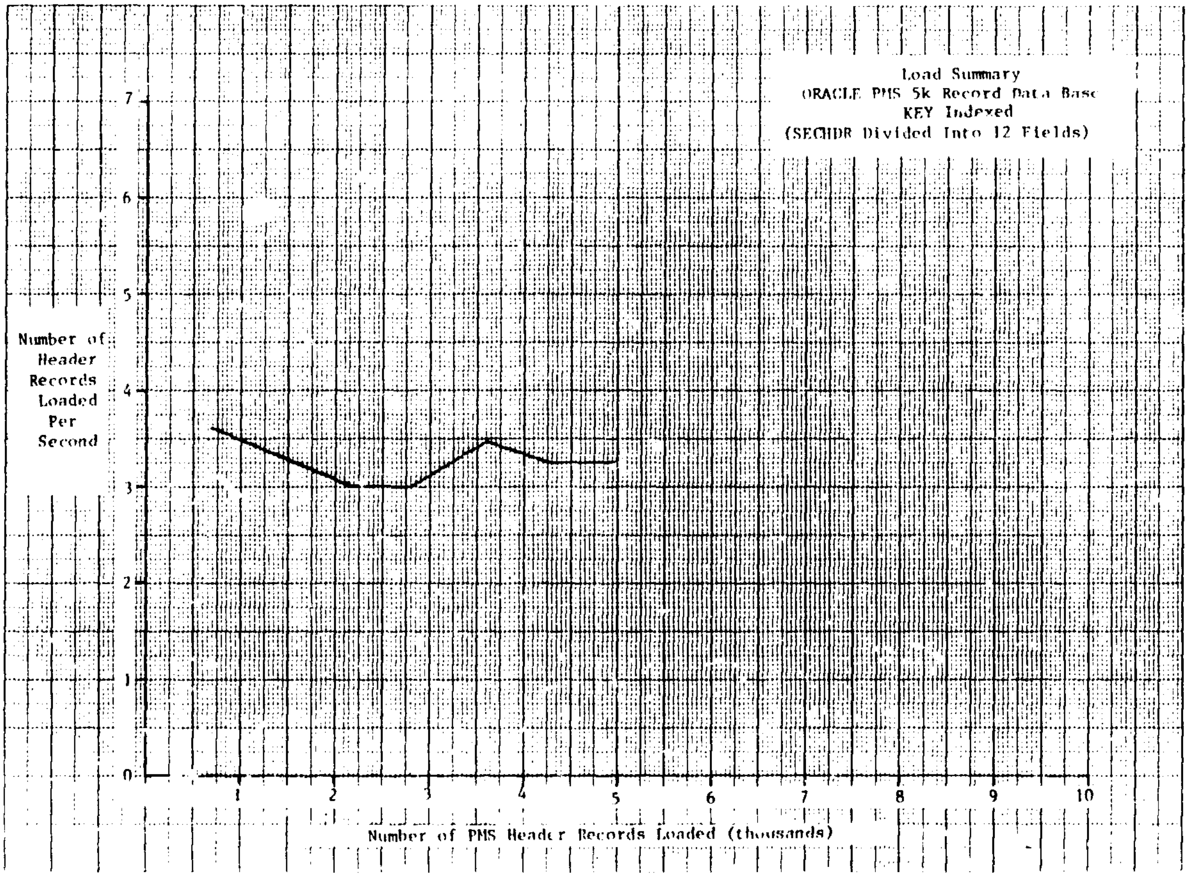
SELECT * FROM HEADER WHERE PRIMARY_KEY = ________

The results of the four data base loads are summarized in the table below.

ORACLE LOAD RESULTS

UIC Description	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
1-64 Byte Character String	10.68	---	468.	360.	3,076.	9,232.
4-16 Byte Character Strings	9.12	+ 14.6	548.	433.	3,071.	8,982.
8-8 Byte Character Strings	7.63	+ 28.6	655.	530.	3,073.	10,577.
16-4 Byte Character Strings	5.27	+ 50.7	948.	792.	4,062.	16,011.





ORIGINAL PAGE IS
OF POOR QUALITY

The number of fields present in the record appeared to have severely impacted performance in loading the data base. Whereas the total length of a header packet did not vary among runs, the amount of information to be managed by ORACLE did.

The query rates for the same 4 data bases are shown here.

ORACLE QUERY RESULTS

UIC Description	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
1-64 Byte Character String	.093	---	23.34	15.00	467	3,022.
4-16 Byte Character Strings	.095	+ 2.2	23.64	15.77	456	3,609.
8-8 Byte Character Strings	.095	+ 2.2	23.83	15.92	454	3,515.
16-4 Byte Character Strings	.098	+ 5.4	24.61	16.13	532	3,199.

Whereas a slight degradation appeared in querying as a result of an increased number of fields in a record, the degree of degradation was much smaller than in loading and should not cause great concern.

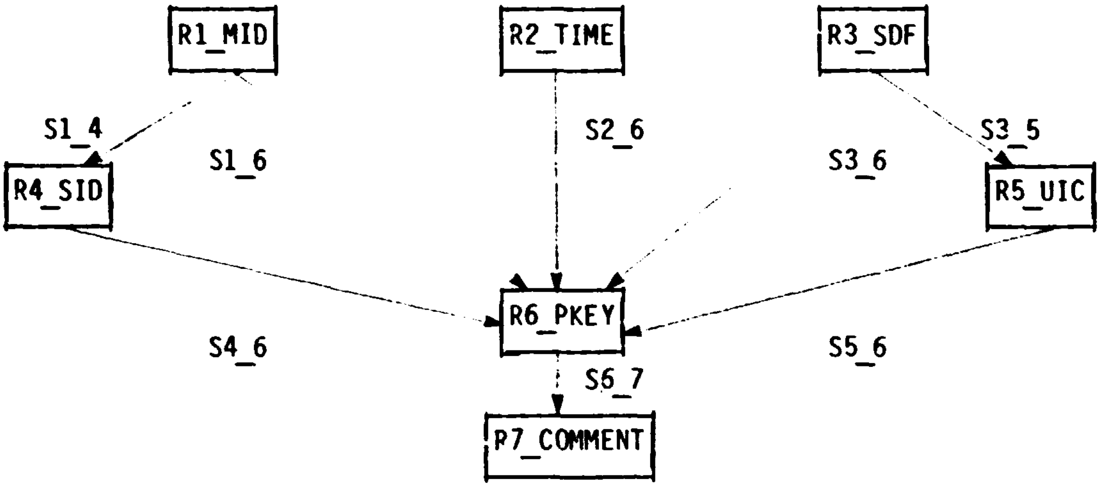
SEED Version B.11.9 Results

This test was run using the original PMS design as referenced in Appendix I.

In the prototype design, shown below, the R6_PKEY record contained the primary header, a time field, a packet length field, and a character field of length 48 (bytes) to hold the remaining secondary header fields. One variation on the design was to alter this 48 byte character field into 3 - 16 byte fields and then again into 6 - 8 byte fields and observe the results to see if the number of fields or field lengths of non-indexed fields impacted load performance. In this test, 10,000 records were loaded each time. As the number of fields grew, the load rates declined slightly, with 7.1 → 4.5 records per second for one field, 6.7 → 3.7 records per second with 3 fields, and 6.1 → 3.5 records per second for 6 fields.

The results are shown on the following page.

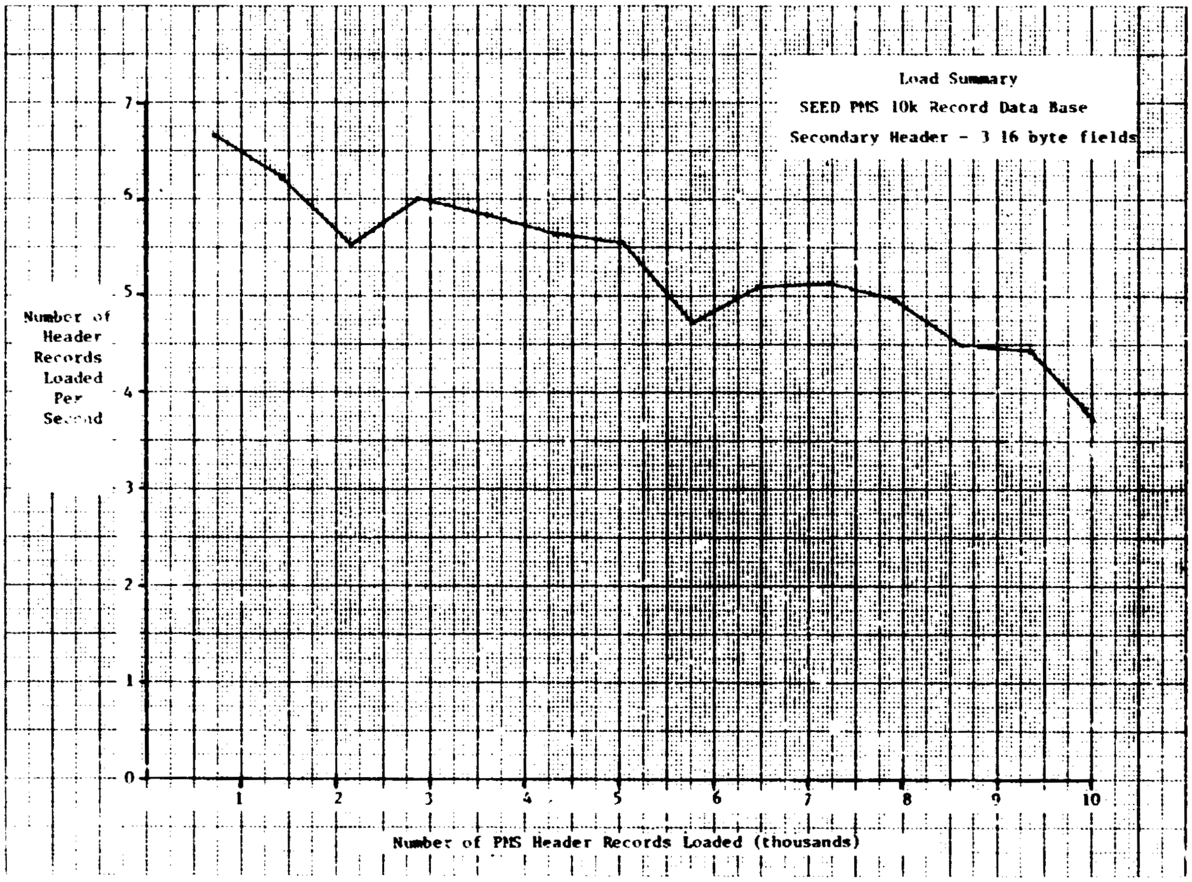
SEED Schema

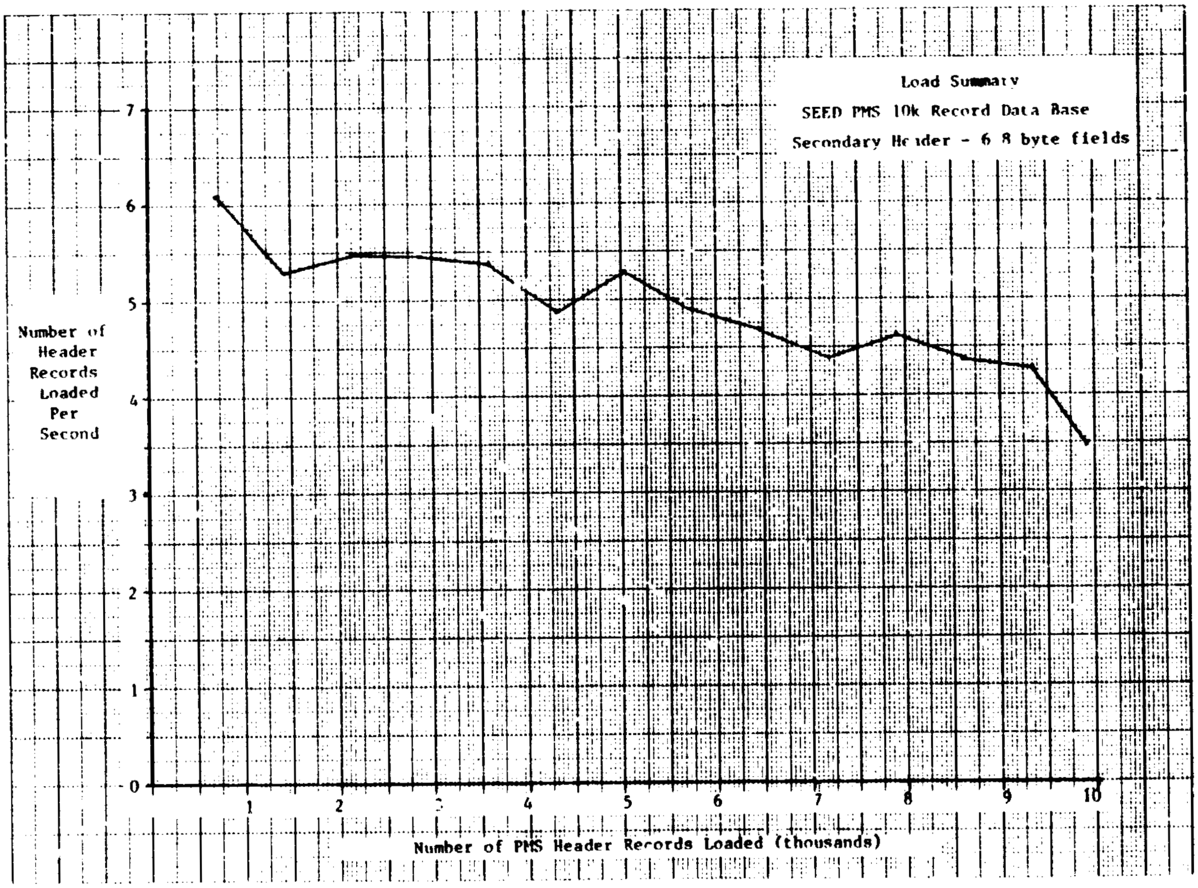


The table below summarizes the results into a more readable form.

SEED LOAD RESULTS

Field Description	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
3-16 Byte Fields	5.69	---
6-8 Byte Fields	5.29	+ 7.0


Load Summary
SEED PMS 10k Record Data Base
Secondary Header - 3 16 byte fields
Number of Header Records Loaded Per Second
0
1
2
3
4
5
6
7
1
2
3
4
5
6
7
8
9
10
Number of PMS Header Records Loaded (thousands)


Load Summary
SEED PMS 10k Record Data Base
Secondary Header - 6 8 byte fields
Number of Header Records Loaded Per Second
0
1
2
3
4
5
6
7
1
2
3
4
5
6
7
8
9
10
Number of PMS Header Records Loaded (thousands)

INGRES Version 1.3 Results

The test design used here was to load 5,000 records into a PMS-like data base with one table, which is diagrammed below.

HEADER Table

MID_SID*	SSC*	SDF*	TIME*	UIC	MESSAGE	HEADER

• Denotes Concatenated Primary Key (Indexed)

The first four fields formed the unique primary key which was indexed after the data was loaded. The UIC field was defined as a 64-byte character string. The data base was then queried 250 times (5% of the records in the data base) on the value of the primary key.

This test procedure was then repeated three times. First, there were 4-16 byte UIC fields, then 8-8 byte fields, and finally 16-4 byte fields. With each data base design the same records were loaded and the same query was performed.

The load results from the four test cases are summarized below as given in the VAX account file. Insertion rates include time necessary to create an index on the primary key as well as loading times.

INGRES LOAD RESULTS

UIC Description	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's
1-64 Byte Character String	9.60	---	521.	271.	13,425
4-16 Byte Character Strings	9.40	+ 2.1	532.	278.	13,432
8-8 Byte Character Strings	9.14	+ 4.8	547.	293.	13,441
16-4 Byte Character Strings	8.98	+ 6.5	557.	308.	13,463

There appears to be a fairly steady degradation in load rates as the number of fields increases.

The results of querying the same four data bases on the primary key value are shown in the following table as reported in the VAX account file. In each case, the query was performed 250 times. The times shown are average retrieval times.

INGRES QUERY RESULTS

UIC Description	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)
1-64 Byte Character String	.43	---	108.
4-16 Byte Character Strings	.46	+ 7.0	116.
8-8 Byte Character Strings	.56	+ 30.2	141.
16-4 Byte Character Strings	.67	+ 55.8	168.

As in the load, the size of a field appears to have an impact on query performance. The impact appears to be much greater, though, in querying than loading.

2.2 Record Structures

DBMS models are designed to provide for the management of multiple data structures. This distinction is one which separates DBMSs from file management systems. If the system can only manage one data structure at a time, it is not a true DBMS for it cannot manage data which is logically related but is organized differently. DBMSs provide this capability in various manners, including the use of different "branch" descriptions in hierarchical systems, different "member" and "owner" descriptions in network systems, and different "table" descriptions in relational systems. In implementing the capability to manage different structures, the DBMS designers have had to design software which recognizes the appropriate structure to use and which interprets how to process that particular structure's data. As a simple means to evaluate whether a more complex data base schema results in decreased performance, a set of tests were defined which altered the conventional PMS-like design used in many of the previous tests into two distinct, identical structures. A PMS header could be inserted into either structure by the load software. The DBMS had to manage a schema description which might be thought of as being twice as complicated as the previous one, which serves as a control for comparison.

The tests were implemented so that 5,000 PMS header records were loaded with approximately half in each structure. Approximately one hundred twenty five headers were queried and retrieved from each of the structures for a total of two hundred fifty queries as was done in earlier tests. A comparison of load and query results obtained with the dual structure to those obtained with a single structure should provide a preliminary basis for estimating if the approaches used for managing complex structures require more processing overhead than do less complex schemas.

ORACLE Version 2.3.2 Results

The data base schema employed for this test was based on one used frequently in previous sections. The table description of that schema is summarized by the following diagram:

HEADER Table

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SDF	MESSAGE	HEADER

*Indexed Field

The schema for the dual structure test has this HEADER table and an identically specified table called HEADER1. The queries used in the test to access the header data were:

```
SELECT * FROM HEADER WHERE PRIMARY_KEY = ____
```

and

```
SELECT * FROM HEADER1 WHERE PRIMARY_KEY = ____
```

The result of the loading is summarized below along with that of the previous single structure control result:

ORACLE LOAD RESULTS

Type of Structure	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
Single Structure (Control)	3.166	---
Dual Structure	2.957	+6.6

The results show a degradation in load performance when the second table is introduced. Closer examination of other statistics reveals something of the cause of the observed difference in insertion rate. Below are included the host and detached process statistics for the control run and dual structure run. The host statistics were obtained from the reporter file and the detached process statistics from the VAX accounting log.

ORACLE LOAD RESULTS

		Control Run	Dual Structure
Host Process	CPU Time (Sec)	121.81	145.11
	Direct I/O's	18	9
	Page Faults	1,848	17,522
Detached Process	CPU Time (Sec)	649.37	677.74
	Direct I/O's	15,921	19,837
	Page Faults	1,246	28,986

The dual structure appears to have significant increases in almost all of the three computer resources, but the page fault increase in both the host and detached processes is far out of proportion. The dramatic increase is attributed to the additional work space and code required to support the multiple "cursor" areas which are required to access both tables. The additional CPU time required in the detached process may be attributable to the page faulting increase and have no direct connection to the data base's complexity.

The results of the querying of 250 values from the control run and of 125 values from each of the tables in the dual structures appear below:

ORACLE QUERY RESULTS

Type of Structure	Average Response Time (Sec)	% Degradation in Average Response Time
Control	.20	---
Dual	.19	- 4.8

These results do not indicate a significant impact in response due to the increased schema complexity. There continues to be a significantly higher number of page faults as evidenced by the following table, where the host process statistics were generated by the reporter and the detached process statistics from the VAX account file.

ORACLE QUERY RESULTS

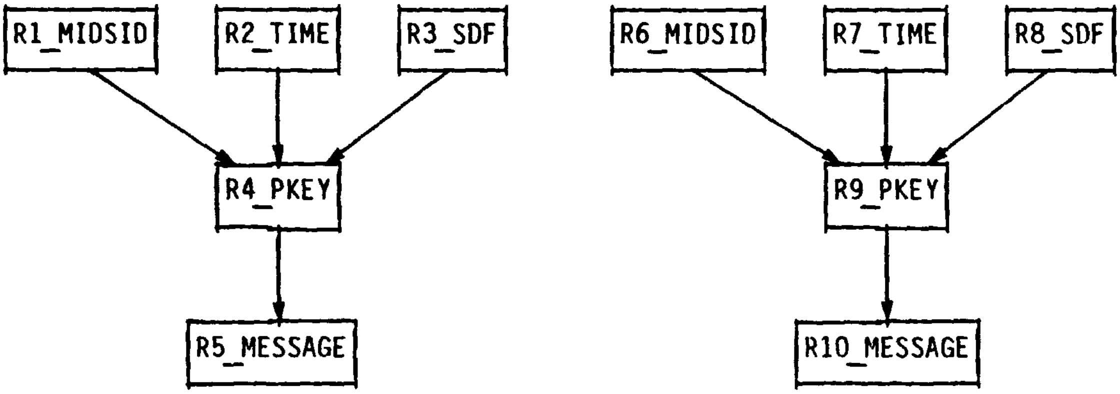
		Control Run	Dual Structure
Host Process	CPU Time (Sec)	7.43	7.91
	Direct I/O's	242	234
	Page Faults	112	1,050
Detached Process	CPU Time (Sec)	25.85	27.71
	Direct I/O's	492	491
	Page Faults	889	2,218

The observed increase in CPU time may again be attributable to the page faults and not to the complexity of the schema. The disproportionate number of page faults is thought to be related to the fact that two cursors are used to perform the queries instead of the one that was used in the control run.

SEED Version C.00.03 Results

The data base design used in this testing is shown in the diagram below. As can be seen from the diagram, the two halves of the schema are identical except for record names. Field names, area names, and set names are also unique, although not shown in this diagram.

SEED Schema



In loading, approximately half of the headers were inserted in each half of the data base structure. The query used in the testing was to form the primary key and "CALC" to the proper record in the proper half of the schema (OBTNC) for 5% of the records in the data base. (See Appendix III)

The loading results are shown in the table below.

SEED LOAD RESULTS

Type of Structure	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
Single Structure (Control)	4.77	---
Dual Structure	3.87	+ 18.9

This table shows that there is a fairly large degradation (about 20%) in loading rates when a significantly more complex schema is introduced.

The results of querying the "double design" data base are shown in the following table, along with the results from the control run:

SEED QUERY RESULTS

Type of Structure	Average Response Time (Sec)	% Degradation in Average Response Time
Single Structure (Control)	.07	---
Dual Structure	.09	+ 28.6

The results show an impact on response time with increased complexity of the schema, probably due to an increase in the number of direct I/O's and page faults. A table showing these values follows:

SEED QUERY RESULTS

Type of Structure	Total Direct I/O's	Total Page Faults
Single Structure (Control)	374	352
Dual Structure	435	664

2.3 Direct Access Alternatives

To retrieve particular records in an efficient manner, DBMSs normally offer a direct access capability. This feature enables a user to specify a value or values for a field or fields which the DBMS can use to locate a record or records which meet the conditions without sequentially searching all the records of that particular type. Alternative approaches used by DBMSs to afford the direct access include hash codes and inverted files. Hash codes process the field or key value and return a physical or logical pointer which identifies the location of the record. Inverted files or B-trees contain the key values and pointers to where the associated records reside. The key values are arranged in order in a hierarchical fashion which facilitates rapid access to any key and its pointer.

Use of a hash code implies direct access in a single or at most two level operation unless different key values generate the same location pointer, giving rise to overflow conditions. This normally results in a chaining of records with duplicate hash values, thus reducing the access efficiency and increasing the insertion overhead. The B-tree approach requires additional space for the inverted file in the data base and as the B-tree grows, slightly more processing is required to navigate additional tree levels. Duplicate key values are usually chained together.

The use of direct access is not without cost, as mentioned above. Insertion, update, and deletion costs can increase when keyed fields are involved and data base space may be consumed. In some cases, the use of keys can have indirect costs. This may occur when, for example, direct access is desired on more than one field in an input data record. Whereas it might be logically desirable to store all of the information from this input data record together (i.e. one table in a relational model or one record type in a network model), the constraints of a particular DBMS may govern that the information be separated because only one field in a table or record type may be accessed directly. Of the DBMSs employed in the testing in this report, SEED Version B.11 is the only one with such a constraint. The following tests measure the efficiency of the direct access approach used as well as the cost of insertion, deletion, and update. They also attempt to measure the overhead associated with key length and duplication of key values.

2.3.1 Direct Access Overhead

To assess the overhead of the direct access techniques used by DBMSs, a set of tests were designed which measure the incremental cost associated with the insertion, update, and deletion of records. The tests were performed on the PMS-like application. A control test was conducted using a data base schema in which no direct access existed for the UIC field. This data base was loaded with 5,000 PMS headers which had unique 16 bit or smaller integer values in the UIC field. Subsequently, one hundred of the records were updated with different UIC values. Next, the same one hundred records which had been modified were deleted. This scenario was then repeated using the direct access technique(s) available in each DBMS package being tested. If multiple techniques for direct access existed in a package, each would undergo a separate set of tests. The results of the "control" tests can be compared with those of the direct access tests to measure the impact of using such features.

ORACLE Version 2.3.2 Results

The table structure used for the control test is summarized in the following figure:

HEADER Table

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SDF	MESSAGE	HEADER

*Indexed Field

The direct access instrument used in ORACLE is a B-tree. The control test did not require an index (B-tree) on the UIC values. The schema was changed in the second test and did require that the UIC field values be referenced in a B-tree. The results of the two loads are summarized in the following table:

ORACLE LOAD RESULTS

Data Base Design	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
UIC Not Indexed	3.16	---
UIC Indexed	2.45	+ 22.5

The addition of the index on the UIC field is substantial, requiring 22.5% more time to load the same number of records (5,000). The cause of the degradation appears to be related to a significant increase in the number of direct I/O operations occurring during the load. In the control load there were about 16,000 direct I/O operations while in the load with the UIC field indexed there were about 28,000, an increase of 75 percent. CPU time also increased by about 18 percent. These figures are shown in the following table and represent the statistics given in the VAX accounting log.

ORACLE LOAD RESULTS

Data Base Design	Total Direct I/O's	Total CPU Time (Sec)
UIC Not Indexed	16,214	786.21
UIC Indexed	28,153	924.16

One hundred PMS headers were then selected to have their UIC field updated. The SQL statement used to perform the update was:

```
UPDATE HEADER SET UIC = -(UIC) WHERE PRIMARY_KEY = _____
```

The results of the update operations are summarized in the table below. The average update time was derived from the reporter file. The total CPU time and total direct I/O's were obtained from the VAX account log, where the corresponding total connect times reported there were 41. and 50. seconds, respectively.

ORACLE UPDATE RESULTS

Data Base Design	Average Update Time (Sec)	% Degradation in Average Update Time	Total CPU Time (Sec)	Total Direct I/O's
UIC Not Indexed	.17	---	22.76	498
UIC Indexed	.26	+ 52.9	26.49	640

The overhead of the index is again obvious, requiring more than 50% additional time to update the UIC field with the index. CPU time increased by 16% and direct I/O's by almost a third as well.

The same one hundred PMS headers updated above were deleted next. The SQL statement used via the host language interface was:

DELETE HEADER WHERE PRIMARY_KEY = _____

The results of the delete tests are summarized in the following table, where the average delete time was taken from the reporter file and the total CPU time and total direct I/O's from the VAX accounting file. In the VAX accounting file, the total connect times were 55. and 56. seconds for the two runs, respectively.

ORACLE DELETE RESULTS

Data Base Design	Average Delete Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's
UIC Not Indexed	.26	27.96	689
UIC Indexed	.26	29.92	689

The results indicate little impact on performance attributable to the presence of the UIC B-tree. The CPU time shows approximately 7% more time required for the data base with UIC indexed but the number of direct I/O operations is identical in the two tests.

ORACLE Version 3.0 Results

In the tests run to determine the impact of applying direct access techniques to a field in a record using ORACLE 3.0, two table structures were defined. The two were identical except for the definition of one field. The table used in the control run, where no index was placed on the UIC field, is shown here.

HEADER Table

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SDF	MESSAGE	HEADER

- Indexed Field

In the other run, an index was placed on UIC in addition to the indices defined above. The old space definition as defined in Section 3.2.1.1.2 of this report, was used in this testing.

First, the data base was loaded with 5,000 PMS-like header records, each with a unique UIC value. The results for the two test cases are given below.

ORACLE LOAD RESULTS

Data Base Design	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
UIC Not Indexed	5.04	---
UIC Indexed	3.52	+ 30.2

A closer examination of the results of the two loads reveals that both the CPU time and number of direct I/O operations increased dramatically when the index was added on UIC. Those results are shown here.

ORACLE LOAD RESULTS

Data Base Design	Total Direct I/O's	Total CPU Time (Sec)
UIC Not Indexed	10,847	661.17
UIC Indexed	2[illegible],756	849.66

Ne:t, one hundred headers were modified by setting the UIC field to its additive inverse. Following this, the same one hundred headers were deleted from the data base. The SQL statements which performed these two functions are given by:

```
UPDATE HEADER SET UIC = - (UIC) WHERE PRIMARY_KEY = ______
```

and

```
DELETE FROM HEADER WHERE PRIMARY_KEY = ______
```

The results of the update operation show that the addition of an index to a field introduces additional overhead in updating as well as loading. The index must also be updated each time the value in the field is updated. This can be seen in the table below.

ORACLE UPDATE RESULTS

Data Base Design	Average Update Time (Sec)	% Degradation in Average Update Time	Total CPU Time (Sec)	Total Direct I/O's
UIC Not Indexed	.10	---	6.95	205
UIC Indexed	.19	+ 90.0	11.76	333

In this case, the addition of the index to the UIC field nearly doubled the time it took to update 100 records.

While the impact of an index on deletion was not as great as on modification, some degradation was apparent, as can be seen from the table below.

ORACLE DELETE RESULTS

Data Base Design	Average Delete Time (Sec)	% Degradation in Average Delete Time	Total CPU Time (Sec)	Total Direct I/O's
UIC Not Indexed	.23	---	14.10	354
UIC Indexed	.26	+ 13.0	16.30	358

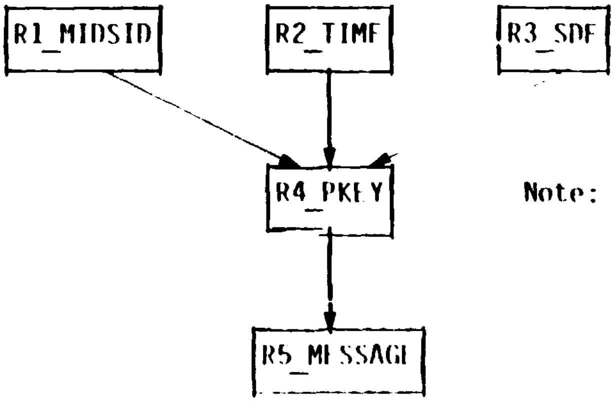
Whereas in a modification of a value which is indexed, the ordering within the index is altered, this is not the case when a value is deleted. The ordering within the index does not vary - only the delete takes place. This could account for the small increase in CPU when UIC was indexed.

SEED Version C.00.03 Results

In SEED, direct access to a particular field may be gained in one of two ways. Either a separate record can be created for the field, where the field is "CALC"ed (hashed), or an index (B-tree) can be created for a non "CALC"ed field in an existing record. The three data base designs for the control test, direct access through record test, and direct access through index test are shown below.

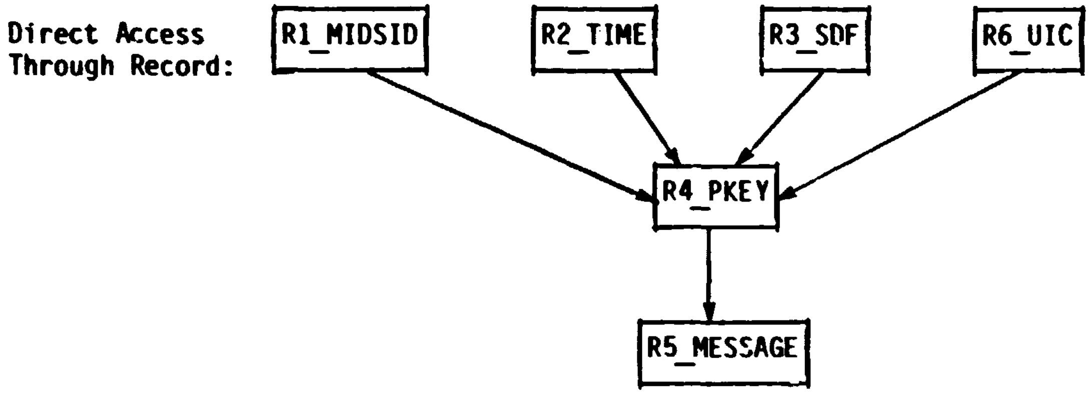
SEED Schema

Control:

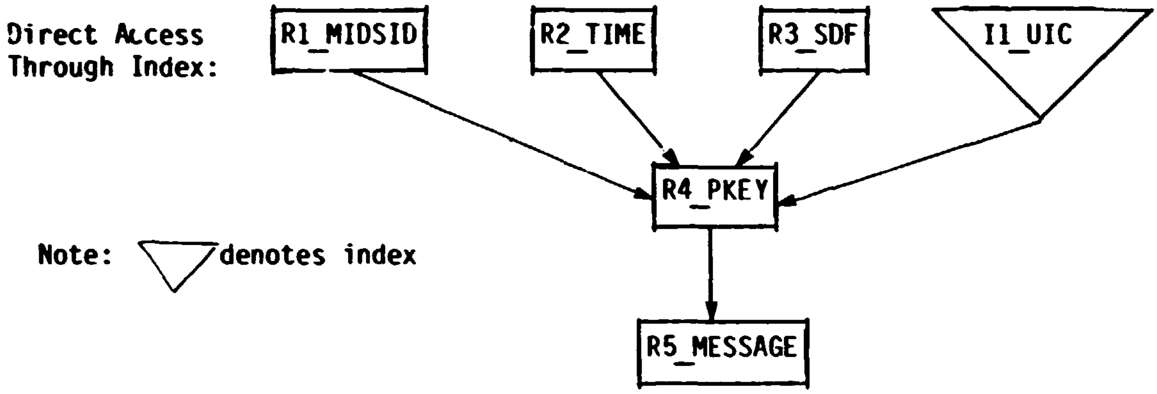


Note: UIC exists as a non-calc field in record R4_PKEY

SEED Schema



SEED Schema



The methods used to alter and delete records in the three test cases were as follows. In the control run, the primary key was formed, correct R4_PKEY record was accessed through the OBTNC command, and the record was either updated or deleted. In the direct access through record test, the correct R6_UIC was accessed through OBTNC, altered or deleted, and its member in R4_PKEY was altered or deleted. In the direct access through index test, the correct R4_PKEY record was located through OBTNI, and then altered or deleted. (See Appendix III)

The loading results of the three runs are shown below.

SEED LOAD RESULTS

Data Base Design	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate	Total Direct I/O's
Control Run	5.00	---	18,286
Direct Access Through Record	2.96	+ 40.8	29,151
Direct Access Through Index	4.44	+ 11.2	18,852

While the time required to load 5,000 records is increased by application of either method of direct access for a field, the percent of degradation is much greater when using a record as opposed to an index, as shown in the third column of the table above. When the R6_UIC record is present in the schema, there is additional overhead for actually loading the R6_UIC record and for forming the proper set linkage connecting it to the R4_PKEY record each time a new input data record is inserted into the data base. Also, because of the additional set linkage, the length of the R4_PKEY record is increased by 4 bytes. The number of SEED pages that are 95-100% full increases from 31 in the control run to 45 in the direct access through record run. The statistics bear out the fact that more direct I/O's are occurring in the second test, increasing from around 18,300 to 29,200, or about 60%. In the direct access through index run, the same number of pages (31) are 95-100% full as in the control run. Also, the number of direct I/O's is only slightly higher, at around 18,900. When using an index, the overhead of placing the value in a B-tree in a separate area of the data base exists, which may account for the small increase in direct I/O's.

The results of altering and deleting 2% of the records in the data base is shown below for each of the three designs.

SEED UPDATE AND DELETE RESULTS

Data Base Design	Average Update Time (Sec)	% Degradation in Average Update Time	Average Delete Time (Sec)	% Degradation in Average Delete Time
Control Run	.08	---	.23	---
Direct Access Through Record	.26	+ 225.0	.30	+ 30.4
Direct Access Through Index	.13	+ 62.5	.28	+ 21.7

In the control run, the UIC field can be updated very quickly because the correct R6_PKEY record is obtained by "CALC"ing on PKEY. After the proper record has been made "current", the UIC field can be updated easily. When a record is created for the field, 2 records must be updated--the R6_UIC which is accessed directly, and the proper R4_PKEY member record. When an index is created for the field, access is slower than a direct hash to the record using the primary key value because of navigating the B-tree, but because of the ordering within the B-tree, access is still faster than through an owner-member access.

The deletion statistics show the same kind of behavior as the modifications. The control run could delete records faster, because only one record is needed to be accessed per delete. Records obtained through indexing could be deleted faster than those obtained through an owner-member relationship (direct access through record), although the difference between the two methods was not as significant as in updating.

In the control runs for the update and delete tests the location of the record in question was determined by "CALC"ing to it using the unique primary key value in the R4_PKEY record. It should be noted that this facility is the reason the control runs can locate the UIC so effectively. If a sequential search of the R4_PKEY records was required to find a particular UIC value the control run would have had much different results.

INGRES Version 1.3 Results

The table structure used for this testing is shown in the diagram below.

HEADER Table

MID_SID	SSC	SDF	TIME	UIC	MESSAGE	HEADER

The two different methods of loading available to INGRES users, copy and repeat append, are described in Section 3.2.1.3.4 of this report. The three data storage structures employed in testing throughout this report - heap, hash, and ISAM are described in Section 3.2.1.3.2 of the report.

In this test, 5,000 records were loaded into the data base using the copy command and were stored in heap structure. Following the load, indices were created for the concatenated primary key (made up of the MID_SID, SDF, SSC, and TIME fields of the input record), and for the MID_SID field and the TIME field. In the first run, the UIC field was defined as I*2 but no index was created for it. Following the load, 2% of the records in the data base, or 100 records, were updated, and then the same 100 records were deleted. In the second run, the UIC field was defined as I*2, and an "ISAM" index was created for the field. As in the first run, 100 records were then updated and deleted.

Because of the manner in which indices are created, the difference in load time between the two runs described above is just the time necessary to create an index on UIC.

The chart below shows the times necessary to load the data and create the various indices.

INGRES LOAD RESULTS

Data Base Design	Load (Sec)	Create Index on PKEY (Sec)	Create Indices on MIDSID and TIME (Sec)	Create Index on UIC (Sec)	Total (Sec)	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
UIC Not Indexed	176.	160.	125.		461.	10.85	---
UIC Indexed	167.	156.	122.	56.	501.	9.98	+ 8.0

There appears to be an 8% degradation when an index is created on UIC. However, the degradation could be considered much more. If the UIC had been indexed on the first run instead of the second, the totals would have been 517 seconds and 445 seconds, respectively. In this case the degradation would have been about 14%. This is because there was a 3-1/2% difference between the times necessary to complete the same three tasks in each run.

The results of updating and deleting the data base are shown below for both cases. The INGRES statements which performed the updates and deletes were:

```
RANGE OF H IS HEADER
REPLACE H(UIC=-H.UIC) WHERE
          H.MID_SID = __________ AND
          H.SSC     = __________ AND
          H.SDF     = __________ AND
          H.TIME    = __________

DELETE H WHERE
          H.MID_SID = __________ AND
          H.SSC     = __________ AND
          H.SDF     = __________ AND
          H.TIME    = __________
```

INGRES UPDATE AND DELETE RESULTS

Data Base Design	Average Update Time (Sec)	% Degradation in Average Update Time	Average Delete Time (Sec)	% Degradation in Average Delete Time
UIC Not Indexed	.79	---	1.04	---
UIC Indexed	1.04	+ 31.6	1.25	+ 20.2

The degradation is significant in both the update and the delete runs, being about 32% and 20%, respectively.

2.3.2 Key Length

The ability to afford direct access to a record based on a particular index value requires that the DBMS software be capable of managing different key value lengths. Character string data is the most likely candidate to be variable in length and has been the target of these tests. Some DBMS implementations have failed to manage more than the first portion of the key value as the index and treat values that are not the same but whose initial portion is identical as duplicates. Other implementations have provided key compression techniques which effectively reduce the required storage needs for managing long key values. The tests designed to investigate this aspect of performance used the PMS-like application used previously (see Appendix I). The UIC field was defined as a character type field. In the initial test, four byte character string values were used, in the second test eight byte values were used, and in the third test twelve byte values were applied. In each case, 5,000 PMS headers were inserted into the data base and 250 headers were accessed using appropriate UIC values as the guiding selection criterion.

ORACLE Version 2.3.2 Results

Since ORACLE permits variable length fields, the table structure used for all the tests was the same. The values loaded into the UIC field determine the character length, thus eliminating the need to define different schemas for each test. The design used specified the UIC field as an "imaged" (indexed) field that was character type data with a maximum length of 12 characters. The table structure for the tests follows:

HEADER Table

PRIMARY_KEY*	MID_SID*	TIME*	UIC*	SDF	MESSAGE	HEADER

*Indexed Field

The results of loading 5,000 PMS headers into the three data bases is summarized below:

ORACLE LOAD RESULTS

UIC Description	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
4 Byte Character String	2.45	---
8 Byte Character String	2.42	+ 1.2
12 Byte Character String	2.45	---

The results of the tests show that no significant difference in performance exists due to the variable character lengths used in these tests.

A single query statement was constructed to exercise each of the data bases once loaded. The format of the SQL statement used is:

SELECT * FROM HEADER WHERE UIC = ___

The query was executed using 250 different UIC values for each data base. The results of the tests are summarized in the following table:

ORACLE QUERY RESULTS

UIC Description	Average Response Time (Sec)	% Degradation in Average Response Time
4 Byte Character String	.19	---
8 Byte Character String	.19	---
12 Byte Character String	.20	+ 5.3

While the 5.3% degradation in the average response time for the 12 byte character string is small and therefore not conclusive evidence of the impact of key length on performance, the trend is present and should be noted.

Seed Version C.00.03 Results

As in the previous section, direct access to a field in a SEED data base may be gained through the use of either a record or an index. For each of the three cases, i.e. Character*4, Character*8, and Character*12, both methods were tested. The data base designs are identical to the designs in the previous section, with the exception that the UIC field was defined as one of the above character lengths.

In the tests which employed the use of records to gain direct access to a UIC field, the query that was performed was to "CALC" (OBTNC) the proper R6_UIC, and then find (OBTNPO) the correct R4_PKEY member on 5% of the records in the data base. In the cases using an index on the UIC field in R4_PKEY, the method used was to do a "find on index" (OBTNI) tc access the proper R4_PKEY record on 5% of the records in the data base. (See Appendix III)

The loading results of all tests are shown below, grouped by method of direct access.

SEED LOAD RESULTS

Method	UIC Description	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate	Total Connect Time (Sec)	Total CPU Time (Sec)	% of Total Connect Devoted To CPU
Record	4 Byte Character String	2.71	---	1,842.89	942.48	51
	8 Byte Character String	3.06	- 12.9	1,633.14	937.44	57
	12 Byte Character String	2.63	+ 3.0	1,899.80	952.04	50
Index	4 Byte Character String	4.41	---	1,134.36	647.93	57
	8 Byte Character String	4.13	+ 6.3	1,210.40	679.40	56
	12 Byte Character String	4.09	+ 7.3	1,223.03	686.53	56

In the "Record" group, the 8 Byte Character String run stands out as having much better performance than either the 4 Byte Character String or 12 Byte Character String run. However, by examing the total CPU times, there is only about 1.5% maximum difference (15 seconds) over the approximately 900 total seconds, a difference which is not significant. However, in the 8 Byte Character String run, a greater percentage of the total connect time was devoted to the CPU than in the other runs, thereby reducing the total time. If, in the 8 Byte Character String run, 50% of the total connect time had been devoted to CPU, as in the other runs, the total connect time would have been about 1,875 seconds, which would have placed it between the 4 Byte Character String and 12 Byte Character String runs, as might be expected.

In the "Index" group, where the percentage of total time devoted to CPU is approximately equal for all three runs, the total connect times show that load rates are impacted by the length of the variable which is to be directly accessed.

In both the "Record" group and "Index" group, the degradations are not dramatic, but the trend does appear and should be noted.

The results of querying on the same data bases are shown below. The results in this table are also grouped by method of direct access.

SEED QUERY RESULTS

Method	UIC Description	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)
Record	4 Byte Character String	.12	---	29.46	13.43
	8 Byte Character String	.11	- 8.3	26.71	13.70
	12 Byte Character String	.12	---	29.93	13.66
Index	4 Byte Character String	.11	---	27.33	13.99
	8 Byte Character String	.11	---	28.48	14.57
	12 Byte Character String	.11	---	28.10	14.30

The total CPU times of all runs are very similar. The length of the direct access field does not appear to significantly impact retrieval rates, regardless of whether the method of direct access is of the "Record" or "Index" type.

2.3.3 Key Duplication

In the preceding section, concern was stated about the effectiveness of some DBMSs in managing larger size field values as indexes. A similar concern exists regarding the effectiveness of the DBMS package in managing duplicate key values. Some direct access implementations have been found to demonstrate significant sensitivities to key duplication. A set of tests were defined which examine DBMS performance in the presence of varying amounts of key duplication. In the first group of tests, four data bases were required that were identical. Into each was loaded the same data with the exception that the data values of one indexed field were unique in the first case, were all duplicated once in the second, were all duplicated five times in the third, and were duplicated ten times in the fourth test.

After loading each data base, a number of records were updated and later deleted to further test the impact of duplication. The PMS-like application was again selected as the test bed (see Appendix I) and the UIC field was chosen to contain the duplicated values (which were integer numbers smaller than 16 bits in length). A total of 5,000 PMS headers were loaded into each data base and 100 records were updated and deleted from each.

In the second group of tests, a character string of length 10 (bytes) was chosen for the key value, and two data bases were loaded. In the first, the first six bytes of the key value were the constant string 'UICUIC' and the last 4 bytes were a unique character string. In the second data base, the first four bytes were a unique character string and the last six bytes were the constant string 'UICUIC'. The purpose of this group of tests was to assess the impact of partial duplication within a character string, as opposed to a numeric value, on performance. As in the first set of tests, 5,000 PMS headers were loaded into each data base and 100 records were updated and deleted from each. This second group of tests were conducted using ORACLE 2.3 and SEED C.0

ORACLE Version 2.3.2 Results

The only variable among the tests was the amount of duplication present in the UIC field. This means that a single data base design was used for all tests. The table structure used is pictorially presented as:

HEADER Table

PRIMARY_KEY*	MID_SID*	TIME*	UIC*	SDF	MESSAGE	HEADER

*Indexed Field

The load results of the first group of four tests are summarized in the following table:

ORACLE LOAD RESULTS

UIC Duplication Factor	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
All Values Unique	2.45	---
All Values Repeated Two Times	2.40	+ 2.0
All Values Repeated Five Times	2.44	+ .4
All Values Repeated Ten Times	2.40	+ 2.0

The results show no evidence to conclude that the presence of duplication adds significantly to the overhead of managing the UIC as an indexed field.

The SQL statements used to test the update and delete functions were, respectively:

UPDATE HEAΓΞR SET UIC = -(UIC) WHERE PRIMARY_KEY = ____

and

DELETE HEADER WHERE PRIMARY_KEY = ___

The results of the update and delete tests are summarized in the table below:

ORACLE UPDATE AND DELETE RESULTS

UIC Duplication Factor	Average Update Time (Sec)	% Degradation in Average Update Time	Average Delete Time (Sec)	% Degradation in Average Delete Time
All Values Unique	.26	---	.26	---
All Values Repeated Two Times	.28	+ 7.7	.27	+ 3.8
All Values Repeated Five Times	.27	+ 3.8	.27	+ 3.8
All Values Repeated Ten Times	.27	+ 3.8	.27	+ 3.8

The results show that no inherent overhead exists due to the implementation of the B-tree management logic chosen in ORACLE that is associated with the presence of duplicate key values.

The second group of loading results are shown in the following table. In both runs, the UIC field was defined as a 10 byte character string. In the first run, the first 6 characters were the String 'UICUIC' and the last 4 characters were unique. In the second run, the first 4 characters were unique and the last 6 characters were the constant string 'UICUIC'.

ORACLE LOAD RESULTS

UIC Description	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
'UICUIC' (variable)	2.41	---
(variable) 'UICUIC'	2.39	+.8

The differences in insertion rate between the two runs appears to be negligible.

The statements which were used to update and delete 2% of the records in the data base are shown here.

```
UPDATE HEADER SET UIC= 'XICUIC _ _ _ _' WHERE PRIMARY_KEY = _______
                            or
                       _ _ _ _ UICUIX'

                            and

          DELETE HEADER WHERE PRIMARY_KEY = _______
```

A table showing the results of updating and deleting records in the two cases follows.

ORACLE UPDATE AND DELETE RESULTS

UIC Description	Average Update Time (Sec)	% Degradation in Average Update Time	Average Delete Time (Sec)	% Degradation in Average Delete Time
'UICUIC' (variable)	.45	---	.27	---
(variable) 'UICUIC'	.44	- 2.2	.29	+ 7.4

While there were differences in modifying and deleting records in the two data bases, the differences were not large enough to draw any definite conclusions about partial duplication in a character string.

ORACLE Version 3.0 Results

The data base design used here was identical to that used in the ORACLE Version 2.3 results presented in the section prior to this. In this testing, the old space definition, as defined in Section 3.2.1.1.2 of this report, was used.

First, a data base was loaded 4 times. The loads were identical except for the amount of repetition present in the UIC field. After each data base was loaded, one hundred headers were modified, and then the same one hundred records were deleted.

The load results are summarized below.

ORACLE LOAD RESULTS

UIC Duplication Factor	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
All Values Unique	3.52	---
All Values Repeated Two Times	3.44	+ 2.3
All Values Repeated Five Times	3.49	+ .9
All Values Repeated Ten Times	3.49	+ .9

The duplication factor did not appear to have any significant effect on the load rates of the four tests.

Likewise, when 100 UIC fields were modified, and when one hundred records were deleted from the data base, no significant variations in rates appeared as a result of duplication in an indexed field. Because the average update times are rounded to two decimal places, the degradation in the second run appears as 10.5%. However, the degradation in total connect time was about 8.8%.

The SQL statements supplied to update the UIC field and delete records from the data base are stated below.

```
UPDATE HEADER SET UIC = - (UIC) WHERE PRIMARY_KEY = _____
```

and

```
DELETE FROM HEADER WHERE PRIMARY_KEY = _____
```

The results of the two tests are shown in the table below.

ORACLE UPDATE AND DELETE RESULTS

UIC Duplication Factor	Average Update Time (Sec)	% Degradation in Average Update Time	Average Delete Time (Sec)
All Values Unique	.19	---	.26
All Values Repeated Two Times	.21	+ 10.5	.26
All Values Repeated Five Times	.20	+ 5.3	.26
All Values Repeated Ten Times	.20	+ 5.3	.26

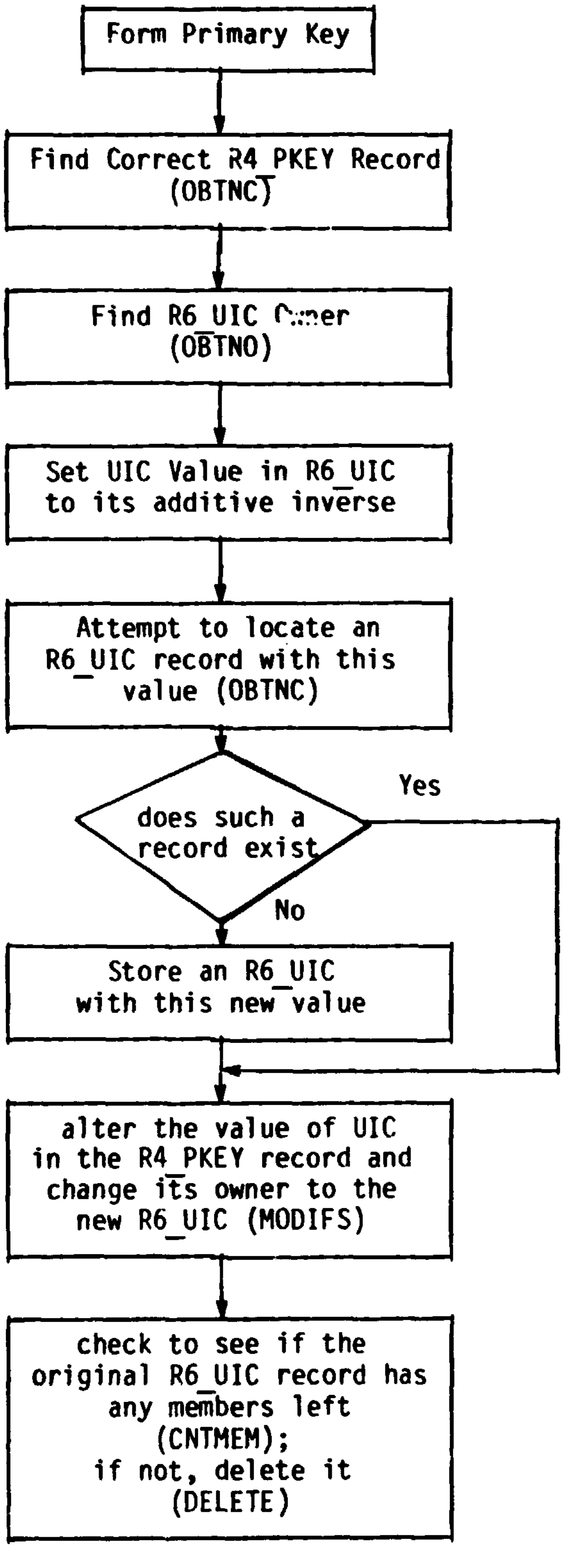
SEED Version C.00.03 Results

As in the two previous sections, this testing was carried out by the use of the two methods of direct access of a variable, creating a record and creating an index. The data base structures were identical to the structures in Section 2.3.1.

In the first group of tests when the method of access was "Record", the method of altering 2% of the records in the data base was defined as described here and as shown in the following flow chart.

First, the primary key was formed and the correct R4_PKEY record was obtained by "CALC"ing on this primary key. Next, tne owner of this record in the set connecting the R6_UIC record and the R4_PKEY record, was found. Because the testing was being performed using the UIC field as the field with duplicates, each unique value of UIC corresponded to one R6_UIC record and many R4_PKEY records. The value of UIC was then altered to its additive inverse in the SEED workspace. Because there was a possibility that one of the previous updates had the same UIC value as the current one, a ch [illegible]de to see whether an R6_UIC record had already been stored with the current additive inverse value. If so, the record was made "current". Otherwise a new R6_UIC record was stored. Next, the value of UIC on the R4_PKEY record was altered to its additive inverse and its owner was changed to the new (or newly obtained) R6_UIC record. Finally, a check was made to see if the original R6_UIC record (before the additive inverse was performed) had any remaining members. If not, the R6_UIC record was deleted.

SEED Flow Diagram


Form Primary Key
Find Correct R4_PKEY Record (OBTNC)
Find R6_UIC Owner (OBTNO)
Set UIC Value in R6_UIC to its additive inverse
Attempt to locate an R6_UIC record with this value (OBTNC)
does such a record exist
Yes
No
Store an R6_UIC with this new value
alter the value of UIC in the R4_PKEY record and change its owner to the new R6_UIC (MODIFS)
check to see if the original R6_UIC record has any members left (CNTMEM); if not, delete it (DELETE)

The deletes were carried out in the 'ollowing way. As in the alter, the primary key was formed, the R4_PKEY record was found, and its owner, the R6_UIC record was found. Then, if the owner had only one member, it was deleted along with the R4_PKEY. If the owner had more than one member, only the R4_PKEY was deleted.

When the method of access was "Index", the alters were performed as follows. The primary key was formed and the correct R4_PKEY record was accessed (OBTNC). The UIC field in R4_PKEY was modified. Any change necessary to update the B-tree was also performed. The delete was very similar. The primary key was formed, the R4_PKEY record was obtained (OBTNC) and deleted. The proper entry in the B-tree was also deleted.

The loading results are shown in the table below.

SEED LOAD RESULTS

Method	UIC Duplication Factor	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate	Total Connect Time (Sec)	Total CPU Time (Sec)
Record	All Values Unique	2.71	---	1,847.	943.
	All Values Repeated Two Times	3.24	- 19.6	1,541.	800.
	All Values Repeated Five Times	2.79	- 3.0	1,790.	737.
	All Values Repeated Ten Times	3.24	- 19.6	1,544.	721.
Index	All Values Unique	4.27	---	1,171.	666.
	All Values Repeated Two Times	4.37	- 2.3	1,145.	661.
	All Values Repeated Five Times	4.31	- .9	1,160.	659.
	All Values Repeated Ten Times	4.24	+ .7	1,179.	673.

When the method of direct access is defined by use of a record, the results show that the CPU time decreases as the duplication factor increases. This is because fewer "owner" records are stored. The correct UIC record is stored only once for a unique value of UIC and then made "current"* for subsequent occurrences of the same value. When the method of direct access is def oy use of an index, the duplication rate does not seem to have a significant impact on load rates.

The results of altering and deleting records in data bases with varying rates of duplication on the UIC field are shown below.

SEED UPDATE AND DELETE RESULTS

Method	UIC Duplication Factor	Average Update Time (Sec)	% Degradation in Average Update Time	Average Delete Time (Sec)	% Degradation in Average Delete Time
Record	All Values Unique	.42	---	.26	---
	All Values Repeated Two Times	.34	- 19.0	.24	- 7.7
	All Values Repeated Five Times	.45	+ 7.1	.26	---
	All Values Repeated Ten Times	.55	+ 31.0	.25	- 3.8
Index	All Values Unique	.13	---	.27	---
	All Values Repeated Two Times	.13	---	.27	---
	All Values Repeated Five Times	.14	+ 7.7	.28	+ 3.7
	All Values Repeated Ten Times	.15	+ 15.4	.28	+ 3.7

While a pattern does exist indicating possible degradation in update performance as the repetition factor of a field increases, in both Record and Index direct access, there is no evidence to support the idea that delete functions are severely affected by the duplication rate of an indexed value. While the delete rates are increasing with increased

* The definition of "current" is most recently accessed.

duplication, the percent of increase is small, indicating possible slight degradation.

In the second group of tests, when the method of access was "Record", the data base was altered by first finding the R4_PKEY record with the correct primary key (OBTNC) and then finding the owner in R6_UIC (OBTNO). The R4_PKEY record was altered. A new R6_UIC record was stored with the modified value and the old value was deleted. In testing the rates for deletion, the proper R4_PKEY and R6_UIC records were obtained as described for altering, and then both records were deleted.

When the method of access was "Index", the updates and deletes were performed as follows. The primary key was formed and the correct R4_PKEY record was accessed (OBTNC). The UIC field in R4_PKEY was modified or deleted. Any change necessary to update the B-tree was also performed.

The loading results are shown below, grouped again by direct access method.

SEED LOAD RESULTS

Method	UIC Description	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate	Total Connect Time (Sec)	Total CPU Time (Sec)	% of Total Connect Devoted to CPU
Record	'UICUIC' (variable)	3.02	---	1,657.	960.	58
	(variable) 'UICUIC'	2.65	+ 12.3	1,889.	964.	51
Index	'UICUIC' (variable)	3.69	---	1,356.	707.	
	(variable) 'UICUIC'	4.07	- 9.8	1,236.	703.	57

While the insertion rate column shows a marked difference in load rates within each pair of numbers, the total CPU times for each pair are very similar (less than 1% difference) and the differences in total connect can be explained by the percentages listed in the last column.

The results of updating and deleting the data bases are given in the tables which follow.

SEED UPDATE RESULTS

Method	UIC Description	Average Update Time (Sec)	% Degradation in Average Update Time
Record	'UICUIC' (variable)	.41	---
	(variable) 'UICUIC'	.42	+ 2.4
Index	'UICUIC' (variable)	.14	---
	(variable) 'UICUIC'	.14	---

SEED DELETE RESULTS

Method	UIC Description	Average Delete Time (Sec)	% Degradation in Average Delete Time	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
Record	'UICUIC' (variable)	.25	---	15.17	374	450
	(variable) 'UICUIC'	.28	+ 12.0	15.48	381	446
Index	'UICUIC' (variable)	.28	---	15.21	441	2,425
	(variable) 'UICUIC'	.32	+ 14.3	17.34	474	4,143

Within each pair of numbers, the average update times do not vary greatly. In each pair of numbers the average delete time is about 12-14% greater for character strings with the variable portion in the leftmost bytes. The total CPU times for the two runs employing "Record" type direct access differ by only about 2% (15.2 and 15.5 seconds, respectively). However, the total CPU times for the "Index" runs show the same type of behavior as average delete time (15.2 and 17.3 seconds). The variation can be explained by comparing the number of direct I/O's and page faults. In the run where the variable portion of the character string is in the rightmost bytes, the number of direct I/O's and page faults are 441 and 2,425, respectively. In the other run, the number of direct I/O's and page faults are 474 and 4,143.

INGRES Version 1.3 Results

In this testing, each of the four data bases (i.e., 5,000 unique values of UIC, 2,500 unique, 1,000 unique, and 500 unique) was loaded using the INGRES copy command. Each was initially loaded as heap data, and indices were then created for the concatenated primary key (MIDSID, SSC, SDF, TIME), MIDSID, TIME, and UIC.

After each data base was loaded, 100 records were modified to their additive inverse, and then the same 100 records were deleted.

The results of loading the four data bases are shown below. The results are broken down into tasks.

INGRES LOAD RESULTS

Data Base Design	Load (Sec)	Create Index on PKEY (Sec)	Create Indices on MIDSID and TIME (Sec)	Create Index on UIC (Sec)	Total (Sec)	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
All Values Unique	167.	156.	122.	56.	501.	9.98	---
All Values Repeated Two Times	174.	166.	125.	58.	523.	9.56	+ 4.2
All Values Repeated Five Times	177.	151.	126.	57.	511.	9.78	+ 2.0
All Values Repeated Ten Times	173.	152.	123.	57.	505.	9.90	+ .8

There is about a 4% maximum difference between the four runs. In the column labeled "Create Index on UIC", the maximum difference of 2 seconds is negligible and can be attributed to variations in the operating system software.

The results of updating and deleting records from the four data bases are shown in the table below.

INGRES UPDATE AND DELETE RESULTS

Data Base Design	Average Update Time (Sec)	% Degradation in Average Update Time	Average Delete Time (Sec)	% Degradation in Average Delete Time
All Values Unique	1.04	---	1.25	---
All Values Repeated Two Times	1.04	---	1.24	- .8
All Values Repeated Five Times	1.06	+ 1.9	1.23	- 1.6
All Values Repeated Ten Times	1.05	+ 1.0	1.22	- 2.4

Again, there does not seem to be any significant degradation in updating or deleting due to uniqueness of the indexed value.

C-2

3.0 LEVEL 2 DATA BASE PERFORMANCE TESTING

The purpose of the tests described in Section 2.0 were to determine the impact of various schema alternatives on DBMS performance. To provide a test bed from which valid comparisons could be made, the test environment was controlled so that the conditions present for one test were as similar as possible to those for another. To assure this, the tests were run in a standalone fashion where the only non-system software requesting resources was the test software (including the DBMS software). The impact of other computer users was not considered. In normal practice, a DBMS must contend with other users for the resources offered by the host computer system. It is also expected that at times multiple users will be simultaneously accessing the data base. One of the purposes of the Level 2 tests was to demonstrate the impact of contention for both system and DBMS resources on data base performance.

A second purpose for the Level 2 test specification was to demonstrate the impact of "tuning" DBMS and computer system parameters that may influence data base performance. The DBMS options available are fully system dependent but can include such things as: data clustering, block size specification, and cache sizes. Therefore, certain tests which were performed using one DBMS may not have a direct counterpart in testing of another DBMS.

All testing in this section was performed using the original PMS design concepts (with the exception of the ORACLE 3.0 testing), which were subsequently altered for the testing in Section 2. In the ORACLE Version 3.0 testing, the data base design was identical to that used in the Section 2 testing. Because the results of this section will be used to determine the kind of degradation that might surface when contention is introduced in a system, it is believed that the data base design or change of design between DBMSs is not as important as is the change in performance within a single DBMS when contention is introduced. See Appendix I for more information on the test environments.

3.1 Contention With Other Users

Contention with other users falls into two categories, contention with DBMS and non-DBMS users. A set of tests were developed to measure the impact on data base performance of each type of contention.

3.1.1 Contention With Non-DBMS Users

To measure the effect of non-DBMS users on the performance of a DBMS, a series of tests were conducted. First, two control runs were made. Each of these runs measured the presence of one function at a time in the computer in standalone mode. In the first control run, 50 queries were made on a PMS-like data base containing 5,000 header packets, and in the second control run, a non-data base related FORTRAN program was compiled. Next, to measure the effect of the FORTRAN compilation on the data base performance, the FORTRAN compilation was performed three times while a 5,000 header PMS-like data base was being loaded. Finally, to measure the effect of an outside process on query performance, a compilation and single query (retrieving 50 packets) were submitted simultaneously, followed by a compilation and 5 query jobs (retrieving 50 packets each) submitted simultaneously.

ORACLE Version 2.3.2 Results

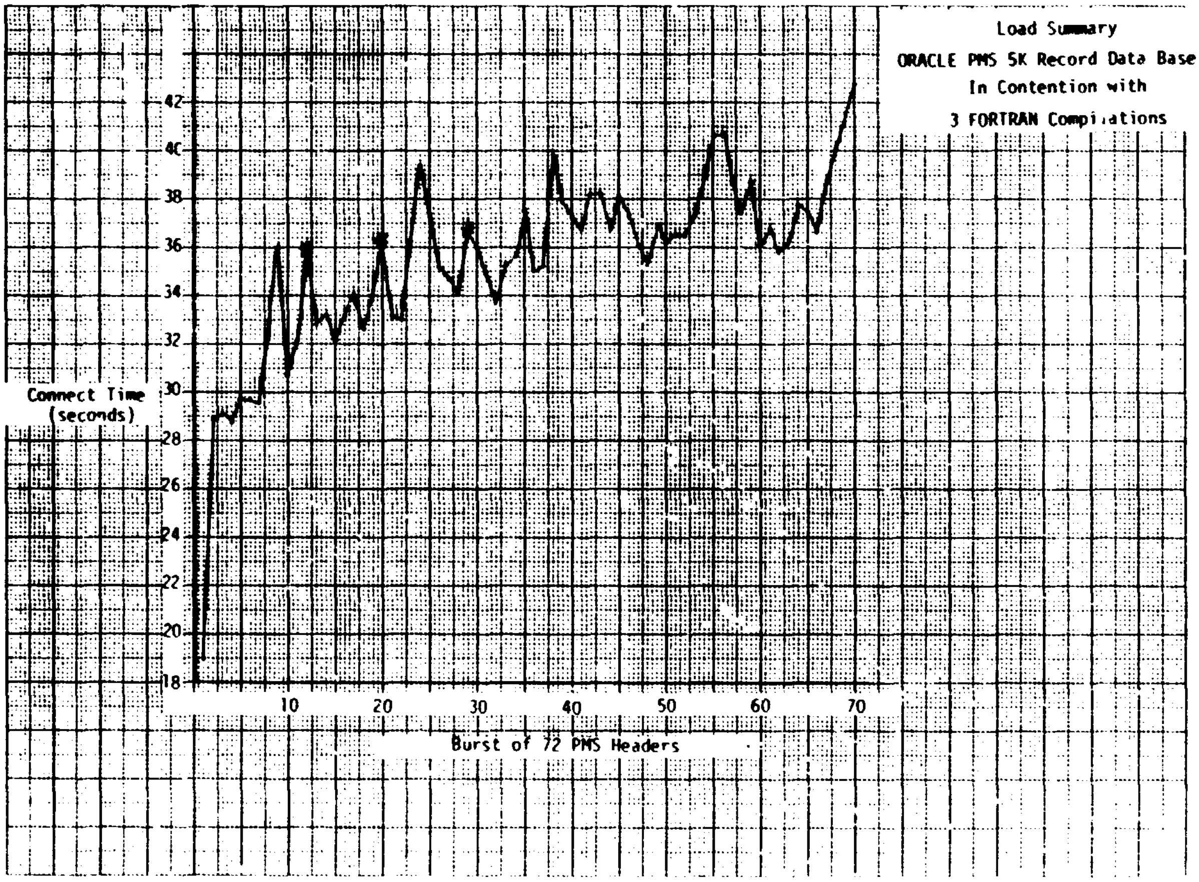
The ORACLE data base used in these tests consisted of the single header table pictured below:

HEADER Table

KEY*	SID*	MID*	SSC	PLP	SDF*	SHID	SIEC	TIME*	PLS	SECHDR	UIC*	COMMENT

* Indexed Field

A job was submitted to load 5,000 PMS-like headers into a data base. As the data base was being loaded, an interactive FORTRAN compilation was submitted three times at approximately 300, 600, and 900 seconds from the start of the load. The graph below summarizes the load performance by plotting the connect time in seconds for each "PMS burst" of seventy-two headers.



The three peaks at bursts 12, 20, and 29 have been starred to identify the occurrence of the FORTRAN compilations. The graph shows that the performance of ORACLE 2.3 in loading the data base was continually degrading as more records were added. So while the starred points do not show a dramatic reduction in performance, it should be noted that each starred point forms a peak, with the point on either side of each star somewhat lower (3-4 seconds). On either side of the peak at the 12th burst, the average insertion rate was about 2.2 headers per second. The rate at the 12th burst was about 2 headers per second. Likewise, on either side of burst 20, the rate was about 2.15 headers per second, while the rate at the 20th burst was about 2 headers per second. Finally, on either side of the 29th burst, the rate was about 2.1 headers per second, but the 29th burst loaded at 1.96 headers per second. It can be concluded that, as would be expected, outside processes do have an impact on data base performance, however slight.

In order to assess the impact of non-DBMS activity in the query environment, the same FORTRAN compilation was used in conjunction with the query routine employed in previous tests that selected 50 rows from the table by qualifying the "WHERE" clause with a unique KEY value (e.g. SELECT * FROM HEADER WHERE KEY = ________). The test consisted of the submission of the FORTRAN compilation along with a single query job and then along with five query jobs.

A summary table of the four runs appears below. In the column titled "Query Times," the connect time is the higher connect time of the job and the detached process as obtained from the VAX account file. The CPU time is the sum of the CPU times for the job and the detached process, also from the VAX account file. The mean time listed under "Query Times" is the average time it took to retrieve 50 headers.

ORACLE QUERY CONTENTION RESULTS

Description	Compile Times		Query Times	
	Total Connect Time (Sec)	Total CPU Time (Sec)	Higher Mean Total Connect Time (Sec)	Sum Mean Total CPU Time (Sec)
Compile Only (no contention)	16.	11.55		
Query Only (no contention)			29.	18.64
Compile & Single Query Run	25.	11.37	44.	19.46
Compile & 5 Query Runs	57.	11.50	120.	17.36

As would be expected, the performance of both the FORTRAN compilation and the ORACLE query suffered as a result of contention for system resources.

ORACLE Version 3.0 Results

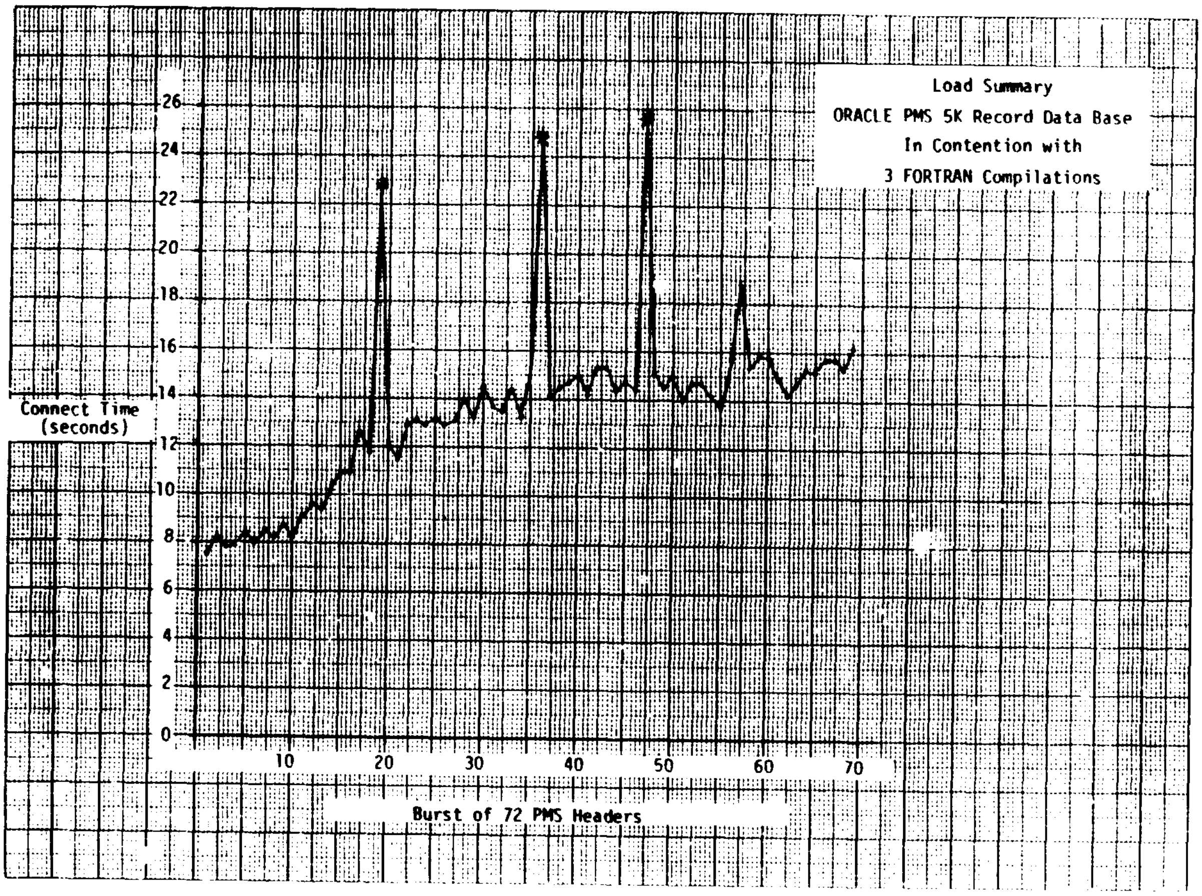
The data base design used in this section was given as:

HEADER Table

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SDF	MESSAGE	HEADER

• Indexed Field

First, a PMS-like data base was loaded with 5,000 header records. While the load was being performed, an interactive FORTRAN program was submitted for compilation three times, at approximately 180, 420, and 600 seconds into the load. In the graph below, a plot of the data base load is shown, giving the total connect time for each burst of 72 header records.



In the graph, the three points where a FORTRAN compilation was submitted are starred. These three points stand out very clearly and show that a data base load can be severely affected by the presence of non-DBMS activity in the system. On either side of the first starred point, the data base was loading at about 6 headers per second, while at that 1st point the rate dropped to about 3.1 headers per second. Likewise, on either side of the 2nd starred point, the load rate was about 5 headers per second and at the 2nd point the rate dropped to about 2.9 headers per second. Finally, on either side of the third starred burst, the rate was about 4.9 headers per second, which dropped to about 2.8 headers per second at the 3rd point.

Next, the same type of test was performed, but on querying of the data base instead of loading. In this test, the same FORTRAN compilation was executed. The data base query that was performed was

SELECT * FROM HEADER WHERE PRIMARY_KEY = _____

The query was performed 50 times. The table below shows the results of four tests. In the first, only the FORTRAN compilation was executed. In the next, only the data base query was performed. Next, a FORTRAN compilation was executed in conjunction with a single data base query, and finally, a FORTRAN compilation was submitted simultaneously with five data base query runs. The statistics were obtained from the VAX account file and each "Query Time" listed is the average time it took to retrieve 50 header packets.

ORACLE QUERY CONTENTION RESULTS

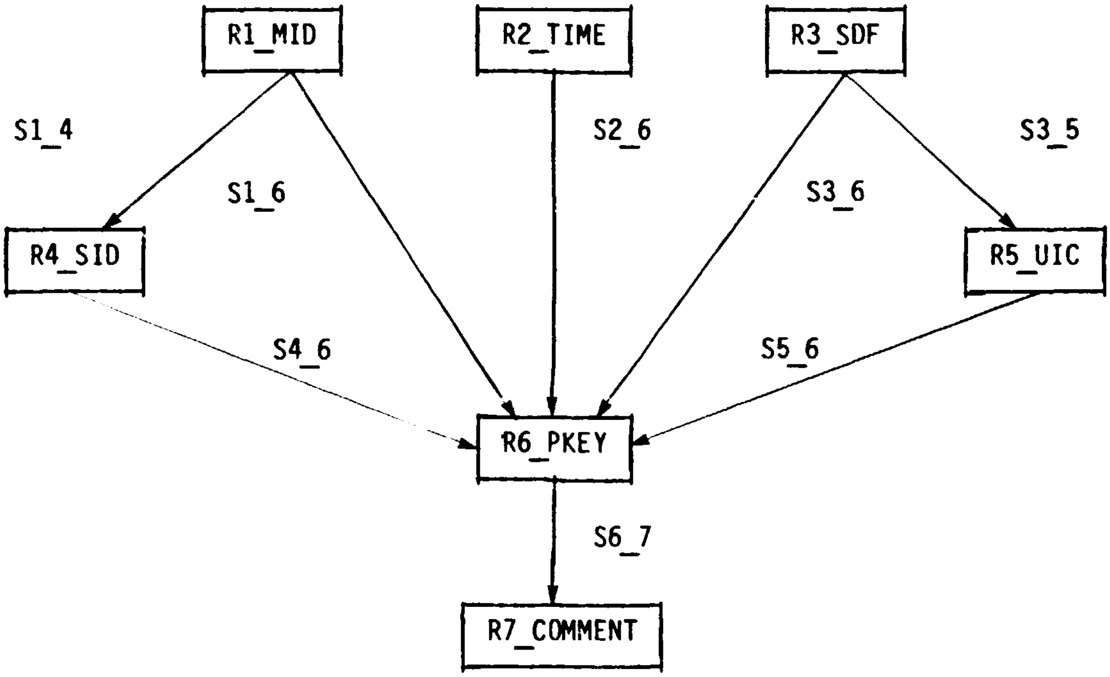
Description	Compile Times		Query Times	
	Total Connect Time (Sec)	Total CPU Time (Sec)	Mean Total Connect Time (Sec)	Mean Total CPU Time (Sec)
Compile Only (no contention)	16.	11.55		
Query Only (no contention)			26.	17.17
Compile & Single Query Run	34.	12.09	39.	17.04
Compile & 5 Query Runs	93.	11.79	108.	16.92

The presence of both a FORTRAN compilation and an ORACLE query (or queries) in the system simultaneously had a significant impact on the performance of both. In a realistic situation, where the contention factor might be much greater, the degree of degradation could be even more significant.

SEED Version B.11.9 Results

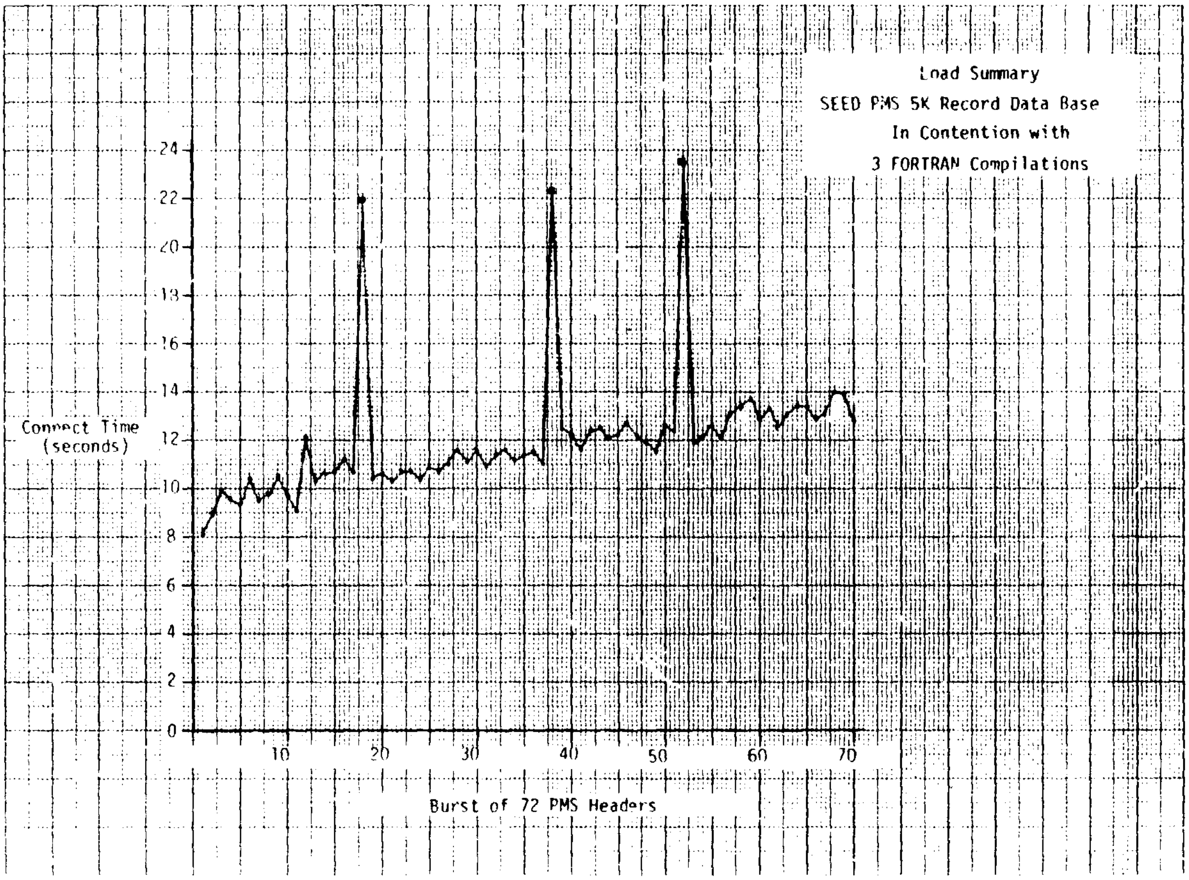
The data base structure that was used in this testing is shown in the diagram below, and an explanation of the structure appears in Appendix I.

SEED Schema



A job was submitted to load 5,000 PMS-like headers into the data base. As the data base was being loaded, at approximately 180, 420, and 600 seconds into the load, an interactive FORTRAN compilation was submitted. A graph showing the loading times for every burst of 72 packets follows. The loading times for the bursts where a FORTRAN compilation appeared are starred and stand out very clearly in the graph.

ORIGINAL PAGE IS
OF POOR QUALITY



The average load rate on either side of the first starred point was about 6.9 headers per second, but dropped to about 3.3 headers per second at the first starred point. On either side of the second starred point, the average insertion rate was about 6.3 headers per second, which dropped to about 3.2 headers per second at that second starred point. Last, the average rate was about 5.9 headers per second on either side of the last starred point and about 3.1 headers per second at that point.

In the table below, the results of a compilation run only, a query run only, a compilation with a single query run, and a compilation with multiple query runs are given. The query that was performed in these runs was to locate 1% or 50 R6 PKEY records, by forming the unique PKEY value and "CALC"ing on it. The statistics were taken from the VAX account log and each query time listed represents the average time for retrieving 50 header packets.

SEED QUERY CONTENTION RESULTS

Description	Compile Times		Query Times	
	Total Connect Time (Sec)	Total CPU Time (Sec)	Mean Total Connect Time (Sec)	Mean Total CPU Time (Sec)
Compile Only (no contention)	16.	11.55		
Query Only (no contention)			18.	12.04
Compile & Single Query Run	29.	11.43	28.	11.41
Compile & 5 Query Runs	74.	11.64	76.	11.69

As can be seen in the chart, the presence of both the compilation and query jobs in the system at the same time severely decreases the performance of each.

INGRES Version 1.3 Results

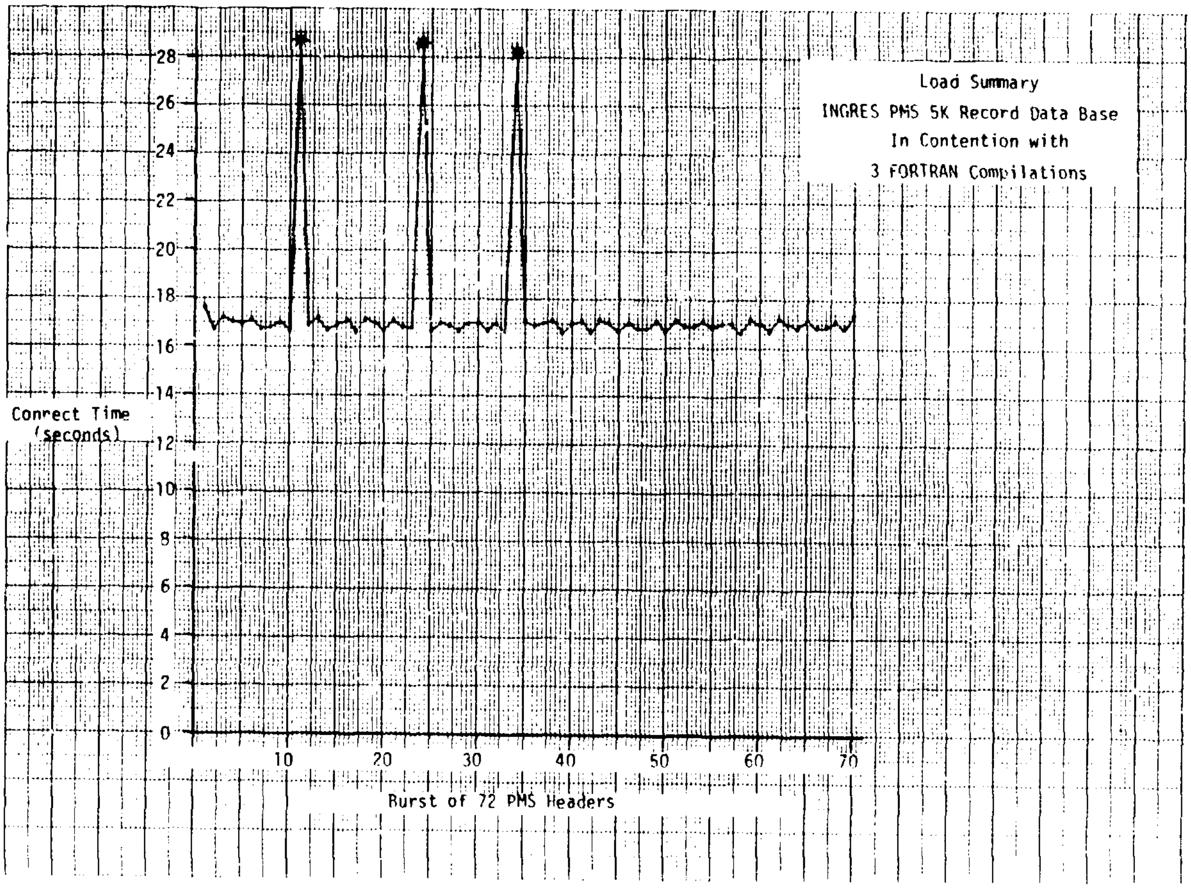
The INGRES data base used in this testing is depicted in the table below.

HEADER Table

MID_SID*	SSC*	SDF*	TIME*	UIC	MESSAGE	HEADER

* Denotes Concatenated Primary Key (Indexed).

Two jobs were submitted to test the impact of contention from a FORTRAN compilation on loading performance. In the first run, 5,000 headers were loaded into a data base with no structure (i.e. HEAP) using repeat append. An interactive FORTRAN compilation was executed at approximately 180, 420, and 600 seconds into the load. In the second run, the data was loaded using the copy command into a data base with no structure. The FORTRAN compilation was executed at approximately 50, 100, and 150 seconds into the load. The explanation for the change in times for submitting the FORTRAN compilations during the "copy" load was that the "copy" load took only 3-4 minutes. If the original set of times had been used, only one FORTRAN compilation would have been completed. For the first load, the graph below shows the load performance by plotting the connect time in seconds for each burst of 72 headers. The following table presents the results of the second load versus the results of a similar load where no contention existed. Because of the nature of the bulk load command, copy, statistics cannot be broken down by burst. Only the total statistics are available.



For the entire data base load (except for the 3 starred points), the average insertion rate was about 4.2 headers per second. That rate dropped to about 2.5 headers per second at each of the 3 starred points.

INGRES LOAD CONTENTION RESULTS

Load Description	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate	Total Connect Time (Sec)
Load using "copy" no FORTRAN compilation	28.4	---	176.
Load using "copy" FORTRAN compilations at 50, 100, 150 seconds	23.7	+ 16.5	211.

It is clear that the impact of outside processes on the loading rate of the INGRES data base is significant. The difference in total connect time between the two runs shown in the table above is 35 seconds. Each compilation used approximately 16 seconds of connect time, or 48 seconds total for the three compilations. In comparing this number with the 35 second time difference between the two loads, it is observed that the data base load utilized only a small portion of the CPU while the compilations were being performed.

The results of a compilation only, a query run only, a compilation with a single query run, and a compilation with 5 query runs are shown in the table below. After the data base had been loaded, a hash was created for the primary key. The query that was performed on 5% of the records in the data base was

```
RANGE OF H IS HEADER
RETRIEVE (H.ALL) WHERE H.MID_SID = _______
                 AND   H.SDF     = _______
                 AND   H.SSC     = _______
                 AND   H.TIME    = _______
```

The statistics shown are from the VAX account log and each time listed under "Query Times" represents an average time for retrieving 50 rows.

INGRES QUERY CONTENTION RESULTS

Description	Compile Times		Query Times	
	Total Connect Time (Sec)	Total CPU Time (Sec)	Higher Mean Total Connect Time (Sec)	Sum Mean Total CPU Time (Sec)
Compile Only (no contention)	16.	11.55		
Query Only (no contention)			36.	24.56
Compile & Single Query Run	29.	11.95	48.	23.10
Compile & 5 Query Runs	77.	11.75	148.4	23.33

Query and compilation performance are both severely affected by the presence of other processes in the system.

3.1.2 Contention With DBMS Users

Because a user may h ve to contend with other DBMS users for resources, it is important to attempt to measure the affect of multiple users on DBMS query performance. This section deals with contention, from both users contending for the same data base and users contending for different data bases.

3.1.2.1 Users Contending for the Same Data Base

In this section, the following tests were devised. First, a set of control runs were made which measured all non-data base related activity in a FORTRAN query program. A set of eleven runs, consisting of from one to ten users and fi.teen users, performed all of the logic which is not DBMS specific. Then a second set of the eleven runs were conducted. This set of runs included the DBMS logic that was excluded in the control set. Results of this set of runs demonstrate the impact of additional users who are contending for similar data base resources.

Comparison of this set's results with those of the control set (between equal numbers of users present) should provide an indication of the amount of influence the DBMS logic has versus that of the remainder of the FORTRAN code present in the tests. Finally, because it is highly unlikely that two users would query the data base at exactly the same time, a set of three runs were conducted, but in this set, instead of submitting the jobs simultaneously, the jobs were submitted at three second intervals. The three runs consisted of 5, 10, and 15 users, respectively.

ORACLE Version 2.3.2 Results

The ORACLE data base was identical to the PMS-like application used in Section 3.1.1. The single table used is summarized below:

HEADER Table

KEY*	SID*	MID*	SSC	PLP	SDF*	SHID	SIEC	TIME*	PLS	SECHDR	UIC*	COMMENT

* Indexed Field

The data base contained 5,000 PMS-like header records, and a single FORTRAN routine which made 50 queries of the following nature was submitted.

SELECT • FROM HEADER WHERE KEY = __________

The FORTRAN query was submitted simultaneously to generate the desired contention levels from which the results below originated.

The table which follows shows the mean connect and CPU time for the control runs and the contention runs when from one to ten, and then 15 users were attempting to access the data base simultaneously. Also reported are the results of the three runs submitted with a three second wait. The contention means are reported as the higher of the mean connect times and the sum of the CPU times for the FORTRAN and detached process runs obtained from the account log. Each "mean" time represents the average time it took to retrieve all 50 rows, not just one row.

ORACLE QUERY CONTENTION RESULTS

Number of Users	Control Run (No db access)		Contention		Contention with 3 Sec. Wait Interval	
	Mean Total Connect Time (Sec)	Mean Total CPU Time (Sec)	Higher Mean Total Connect Time (Sec)	Sum Mean Total CPU Time (Sec)	Higher Mean Total Connect Time (Sec)	Sum Mean Total CPU Time (Sec)
1	12.0	8.4	29.0	18.64		
2	20.5	7.9	46.5	17.48		
3	38.7	8.2	62.0	17.28		
4	40.0	8.0	83.0	17.32		
5	47.7	7.7	108.0	17.20	101.4	17.39
6	60.3	7.8	121.3	17.03		
7	68.9	7.9	155.7	17.44		
8	79.3	7.9	168.3	17.28		
9	88.4	8.0	190.1	17.43		
10	99.2	7.9	209.6	17.59	194.6	17.76
15	137.3	7.8	291.6	17.40	291.0	18.11

These results show a relatively constant increase in connect time proportional to the number of contending jobs present. The average CPU time does not vary significantly as the number of users increases, indicating that no internal contention for ORACLE resources occurs. The degradation in connect time observed when contending jobs are present is what one would expect to see, as demonstrated by the results of the control runs.

Examination of some other statistics revealed an impressive example of the effectiveness of ORACLE's I/O buffering and caching techniques. Below is a table which summarizes the average number of direct I/O operations performed by the detached processes spawned by ORACLE for each active FORTRAN query job submitted.

ORACLE QUERY CONTENTION RESULTS

Number of Users	Average Number of Direct I/O Operations Per Detached Process	Average Number of Buffered I/O Operations Per Detached Process
1	129.0	340.0
2	62.5	373.5
3	46.0	378.0
4	36.0	361.0
5	30.2	381.6
6	26.2	416.5
7	23.6	418.3
8	21.6	427.3
9	21.2	493.3
10	20.4	484.2
15	42.1	574.1

The dramatic reduction in direct I/O's per process is possible because ORACLE recognizes the presence within its own buffers of the logical blocks requested by the detached process. This eliminates the need to initiate a direct read. It should be noted that, since each job in the test is reading the same set of logical blocks, the results above exaggerate the effectiveness of ORACLE's I/O buffering and caching somewhat. The increase in average I/O's at the fifteen user level is evidence of this fact since at this point the operations have begun to lose some of their synchronization resulting in some blocks being read more than once. Also included in the preceding table is the average number of buffered I/O's per each detached process. These operations are primarily related to VAX "mailbox" communications and it is not clear why they are increasing as the number of processes present increases.

ORACLE Version 3.0 Results

In this testing, the ORACLE data base design was identical to that in Section 3. .1, and is shown here.

HEADER Table

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SDF	MESSAGE	HEADER

* Indexed Field

In this scenario, 5,000 PMS-like header records had previously been loaded into the data base. From one to ten, and then 15 users queried the data base, each executing the following query on 1%, or 50 of the records in the data base:

```
SELECT * FROM HEADER WHERE PRIMARY_KEY = ______
```

The table below shows the results of control tests where no data base FORTRAN code was executed followed by the results of runs where the data base FORTRAN code was executed and the jobs were submitted simultaneously, and finally where the runs were submitted in 3 second intervals. In each entry of the table, the results shown are the average of the results obtained for that number of users as reported in the VAX account file. Each "Mean Total Connect Time", for example, represents the average time it took to retrieve all 50 records.

ORACLE QUERY CONTENTION RESULTS

Number of Users	Control Run (No db access)		Contention		Contention with 3 Sec. Wait Interval	
	Mean Total Connect Time (Sec)	Mean Total CPU Time (Sec)	Mean Total Connect Time (Sec)	Mean Total CPU Time (Sec)	Mean Total Connect Time (Sec)	Mean Total CPU Time (Sec)
1	12.0	8.4	26.	17.17		
2	20.5	7.9	45.	16.55		
3	38.7	8.2	62.	17.01		
4	40.0	8.0	81.	17.07		
5	47.7	7.7	101.	16.70	93.	17.14
6	60.3	7.8	120.	16.95		
7	68.9	7.9	139.	16.68		
8	79.3	7.9	157.	16.55		
9	88.4	8.0	174.	16.51		
10	99.2	7.9	197.	16.71	178.	16.96
15	137.3	7.8	283.	16.42	259.	16.83

There is an almost linear increase in total connect time as the number of contending users increases. The CPU time does not vary significantly between runs however, indicating little if any contention for ORACLE resources. Another indication of this is the fact that in all cases, the total connect time in the column headed "Contention" is about 2 to 2.1 times the total connect time in the column headed "Control Run." If there was any contention for data base resources present, this factor would not remain constant. It should also be noted that an improvement in performance occurs when the jobs are not submitted simultaneously.

SEED Version B.11.9 Results

The data base design used in this testing was identical to the design described in Section 3.1.1. The data base was loaded with 5,000 PMS-like records and then 50 queries were performed. In the query, the primary key, PKEY, was formed and the proper record was found by hashing on the PKEY value (OBTNC).

The table below shows the mean total connect time and mean total CPU time for the control runs and contention runs, when from one to ten, and then 15 users were attempting to access the data base simultaneously. Also summarized are contention runs with 5, 10, and 15 users present with a three second interval between the submission of contending runs. The statistics were obtained from the VAX account file. "Mean Total _____" is interpreted as the average time it took to retrieve all 50 records.

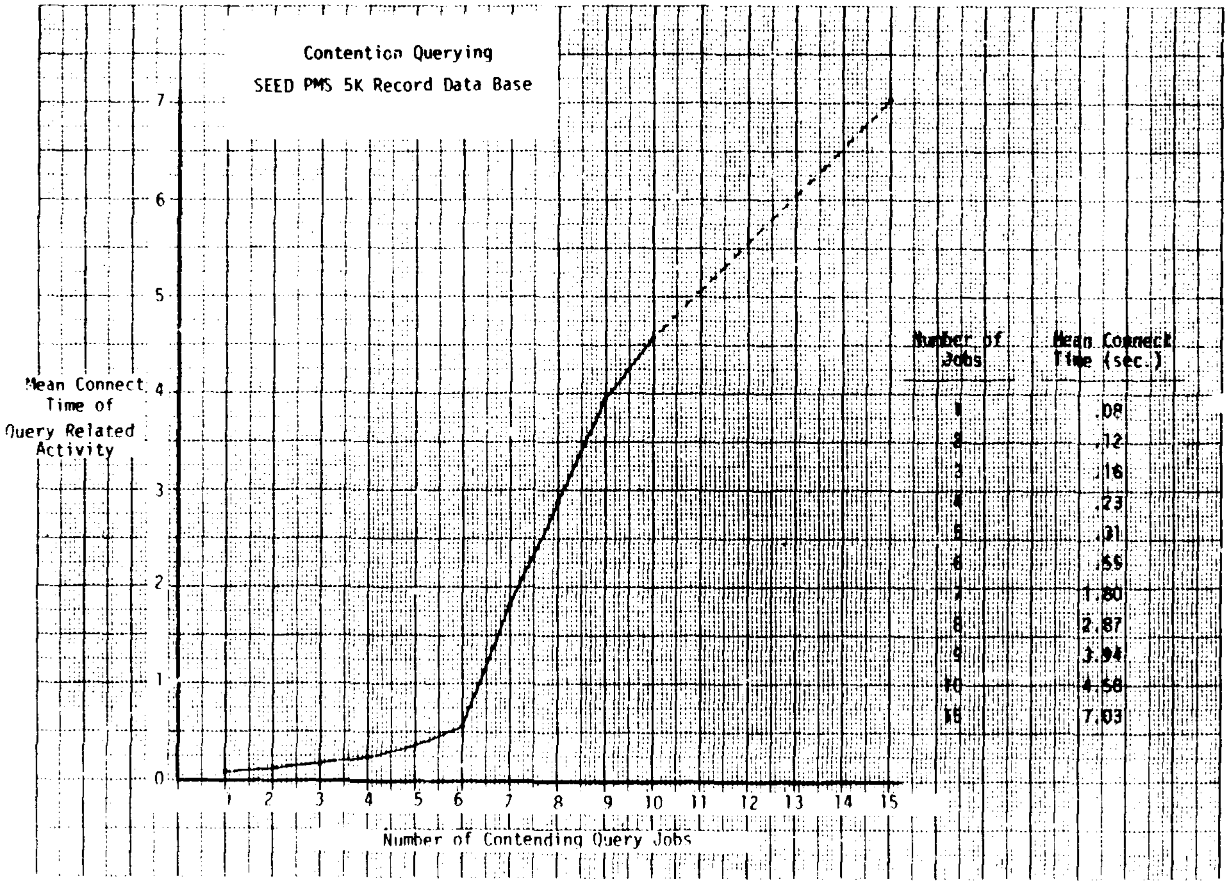
SEED QUERY CONTENTION RESULTS

Number of Users	Control Run (No db access)		Contention		Contention with 3 Sec. Wait Interval	
	Mean Total Connect Time (Sec)	Mean Total CPU Time (Sec)	Mean Total Connect Time (Sec)	Mean Total CPU Time (Sec)	Mean Total Connect Time (Sec)	Mean Total CPU Time (Sec)
1	12.0	8.4	18.0	12.0		
2	20.5	7.9	30.5	11.9		
3	38.7	8.2	45.3	11.9		
4	40.0	8.0	61.0	11.5		
5	47.7	7.7	79.4	12.0	65.6	12.1
6	60.3	7.8	97.7	12.8		
7	68.9	7.9	170.9	14.0		
8	79.3	7.9	250.0	14.6		
9	88.4	8.0	302.9	14.7		
10	99.2	7.9	367.6	14.8	122.9	12.4
15	137.3	7.8	562.3	14.6	359.1	13.8

While the mean CPU time continued to be fairly constant throughout the control runs, it increased as the number of users increased in the contention runs. The increase in mean CPU time might be attributed to the fact that when a job is running and an interrupt occurs, the overhead of processing that interrupt is charged to the job. As the number of jobs trying to run simultaneously increases, the number of interrupts increases and the likelihood that a job is actively running when an interrupt occurs (rather than a system routine) increases, thus the increase in mean CPU time. The biggest jump in mean total connect time takes place between 6 and 7 users. It is most likely that, at this point, in addition to the extra overhead of processing interrupts, the virtual system began to swap jobs in and out of memory more frequently because of a lack of main memory partition space available to satisfy all the jobs present.

By submitting the query jobs at 3 second intervals, which represents a more realistic scenario, the contention for resources was reduced significantly.

In the following graph, the mean connect time of query related activity is plotted versus the number of jobs contending for resources. The mean connect time is the average time necessary to retrieve one R6_PKEY record. As in the previous table, a large jump appears between 6 and 7 users.



Number of Jobs	Mean Connect Time (sec.)
1	.08
2	.12
3	.16
4	.23
5	.31
6	.55
7	1.80
8	2.87
9	3.94
10	4.58
15	7.03

INGRES Version 1.3 Results

The INGRES data base used in this testing was identical to that used in Section 3.1.1. The data base contained a single table, HEADER, which is described in the diagram below.

HEADER Table

MID_SID*	SSC*	SDF*	TIME*	UIC	MESSAGE	HEADER

* Denotes Concatenated Primary Key (Indexed)

The table was stored in hash structure with the hash being on the multi-field key consisting of MID_SID, SDF, SSC, and TIME. The query that was performed (on 50 records) to assess the impact of multiple users accessing the data base simultaneously was:

```
RANGE OF H IS HEADER
RETRIEVE (H.ALL) WHERE  H.MID_SID = _____
                 AND    H.SDF     = _____
                 AND    H.SSC     = _____
                 AND    H.TIME    = _____
```

The table below is a comparison of the control runs (i.e. no data base access), simultaneous contention querying, and contention querying with a 3 second wait. The statistics were generated from the VAX account log. The "means" reported represent the average time it took to retrieve all 50 rows.

INGRES QUERY CONTENTION RESULTS

Number of Users	Control Run (No db access)		Contention		Contention with 3 Sec. Wait Interval	
	Mean Total Connect Time (Sec)	Mean Total CPU Time (Sec)	Higher Mean Total Connect Time (Sec)	Sum Mean Total CPU Time (Sec)	Higher Mean Total Connect Time (Sec)	Sum Mean Total CPU Time (Sec)
1	12.0	8.4	33	23.48		
2	20.5	7.9	58.5	23.24		
3	38.7	8.2	85.7	23.37		
4	40.0	8.0	113.3	23.38		
5	47.7	7.7	141.4	23.35	132.2	23.17
6	60.3	7.8	171.2	23.54		
7	68.9	7.9	209.	23.54		
8	79.3	7.9	227.3	23.48		
9	88.4	8.0	258.9	23.44		
10	99.2	7.9	284.	23.48	264.6	23.42
15	137.3	7.8	425.6	23.55	384.7	23.50

While the CPU times remained relatively constant throughout the entire scenario of runs, indicating no contention for INGRES resources, the connect times increased proportionately to the number of contending jobs present. This was to be expected. The difference in total connect time between jobs submitted simultaneously and those submitted with a 3 second wait interval is significant.

Another interesting statistic which appeared was the number of direct I/O's issued per user for contending users. These are shown in the following table.

INGRES QUERY CONTENTION RESULTS

Number of Users	Average Number of Direct I/O Operations Per Detached Process	
	Simultaneous Submits	3 Sec. Wait Interval
1	61.	
2	35.	
3	26.3	
4	22.	
5	19.4	32.
6	17.7	
7	16.4	
8	15.5	
9	14.8	
10	14.2	24.1
15	13.	34.7

Each of the contending jobs reads the same set of records from the data base. This table suggests that INGRES can recognize that certain logical blocks are already in its buffers and eliminate a direct I/O. This can be seen more clearly by comparing the two columns in the table. When the jobs are submitted simultaneously, there is a greater chance that the logical block requested by one process will still be in a buffer from a previous process. In contrast, where there is a 3 second wait between jobs, there is a higher likelihood that by the time a job requests a certain logical block, the buffer where that block may have resided has been replaced with a new record.

3.1.2.2 Users Contending for Different Data Bases

This section was included to assess the impact on a user's process when other users are contending for data base resources, either of the same or different data bases.

In this test, varying numbers of PCDB queries and PMS queries were submitted simultaneously. The PCDB query which was performed accessed the TAPEID, ARCHIVER, CAT, and CATEGORY for each tape in the data base. The PMS query accessed 5%, or 250, of the header records in the data base.

In each test, up to 5 queries were submitted. The test was conducted using the ORACLE and SEED DBMSs.

ORACLE Version 2.3.2 Results

The original PMS design was used for this testing. Both this design and the PCDB design are discussed in Appendices I and II, respectively.

In this test, varying numbers of PCDB queries and PMS queries were submitted simultaneously. The queries chosen for the PCDB and PMS applications were:

```
PCDB -  SELECT TAPE.TAPEID, ARCHIVER, CAT, CATEGORY
        FROM TAPE, CAT
        WHERE TAPE.TAPEID= <'_____'>
        AND TAPE.TAPEID=CAT.TAPEID

PMS  -  SELECT * FROM HEADER WHERE KEY = '_____'
```

In the PCDB query, the information was retrieved for each of 55 tapes in the data base. In the PMS query, the information was retrieved for 5% (or 250) of the records in the data base.

In each test, up to 5 queries were submitted. The results are shown in tne table on the next page and were obtained from the VAX account file. The numbers across the top of the page represent the job number within a test. The tests are divided by the horizontal lines. Within a test, the term PCDB or PMS is placed next to the results, indicating which query was executed. For example, the first test consisted of one PCDB query and the last test consisted of two PCDB queries and three PMS queries. For each test, the results include both the host job and the detached process. The connect time given is the higher of that for the job and the detached process. The CPU time is the sum of the CPU time for the host job and the detached process.

The PCDB query results indicate that response time was more dependent on the total number of jobs in the system rather than the number of PCDB jobs in the system. Looking at rows 1, 7, 11, 12, and 13, where there is one PCDB job in each, the connect times increased as the total number of jobs increased. Looking at rows 2 and 7, the presence of a PMS job impacted the results of the PCDB query more than the presence of 2 PCDB queries in the system.

Likewise, the PMS query results were more dependent on the number of PMS jobs in the system rather than the total number of jobs. Looking at rows 6-10, the PMS connect times ranged from 78 seconds to 138 seconds. In each job there was one PMS query present. In row 11, two PMS jobs were introduced. The connect time for each was abcut the same as row 10, where there were 5 jobs, but only 1 PMS job. In row 8, where there were 3 jobs present, but only one PMS job, the connect time for that PMS job was significantly lower than in row 11.

This seems to indicate a much higher contention for resources when PMS jobs were introduced. It should also be noted that when one PCDB query was present, the CPU time was 54% of the total connect time, and when one PMS query was present, the CPU time was 64% of the total connect time.

ORACLE-PCDB and PMS Queries

Job Test	1			2			3			4			5		
	Connect Time (Sec)		CPU Time (Sec)	Connect Time (Sec)		CPU Time (Sec)	Connect Time (Sec)		CPU Time (Sec)	Connect Time (Sec)		CPU Time (Sec)	Connect Time (Sec)		CPU Time (Sec)
1	26.	PCDB	13.72												
2	34.	PCDB	12.82	32.	PCDB	10.35									
3	53.	PCDB	12.36	48.	PCDB	11.40	49.	PCDB	12.69						
4	61.	PCDB	11.98	65.	PCDB	11.42	64.	PCDB	12.52	63.	PCDB	11.26			
5	85.	PCDB	14.55	82.	PCDB	12.03	82.	PCDB	12.35	79.	PCDB	12.37	74.	PCDB	11.18
6	78.	PMS	50.12												
7	45.	PCDB	13.92	88.	PMS	48.97									
8	57.	PCDB	12.55	54.	PCDB	12.75	105.	PMS	49.91						
9	72.	PCDB	14.15	74.	PCDB	11.53	68.	PCDB	11.58	122.	PMS	49.88			
10	91.	PCDB	12.77	93.	PCDB	11.73	89.	PCDB	13.42	84.	PCDB	11.19	138.	PMS	49.84
11	57.	PCDB	13.91	140.	PMS	47.85	136.	PMS	48.57						
12	69.	PCDB	13.69	191.	PMS	46.21	189.	PMS	48.92	188.	PMS	46.59			
13	90.	PCDB	13.80	248.	PMS	47.17	246.	PMS	49.32	249.	PMS	48.74	251.	PMS	48.21
14	77.	PCDB	13.50	75.	PCDB	11.68	157.	PMS	47.88	153.	PMS	49.54			
15	86.	PCDB	11.64	92.	PCDB	12.97	91.	PCDB	11.62	174.	PMS	49.04	169.	PMS	48.95
16	90	PCDB	13.71	82.	PCDB	10.91	212.	PMS	49.79	209.	PMS	48.47	209.	PMS	48.18

SEED Version B.11.9 Results

The data base designs for both the PMS application and the PCDB application appear in Appendices I and II, respectively. In this testing. the original PMS design was used.

In this scenario of runs, varying numbers of PMS-like queries and PCDB queries were submitted simultaneously. In each test, a maximum of 5 queries were submitted. The following chart shows the results of each test, as reported in the VAX accounting log. The horizontal line divisions separate tests. The numbers across the top of the chart stand for job 1, 2, 3, 4, and 5 within the test. Above the job is a specification of whether the job was a PMS or PCDB job. For example, test one consisted of one PCDB query and test two - 2 PCDB queries. Test 7 consisted of one PCDB query and one PMS query submitted simultaneously, and so on.

One PCDB query consisted of retrieving the TAPEID, ARCHIVER, CAT, and CATEGORY for each of the 55 tapes in the data base, by "CALC"ing on the TAPEID.

One PMS-like query consisted of retrieving a PMS header after computing the PKEY value from the input record and "CALC"ing on it. This was repeated for 5% (or 250) of the records in the data base.

While the CPU times for the PCDB queries remain constant, and the CPU times for the PMS queries remain constant (though much greater than the PCDB queires), the connect times increase as more jobs are in the system. The connect time for the PCDB runs is more dependent on the total number of jobs in the system than the number of PCDB jobs in the system. The connect times for the PMS runs are dependent on the total number of jobs in the system but also on the number of PMS jobs within the total number. This tends to indicate a greater contention for data base resources in the PMS query than in the PCDB query.

SFED-PCDB and PMS Queries

TEST \ JOB	1		2		3		4		5	
	Connect Time (Sec)	CPU Time (Sec)	Connect Time (Sec)	CPU Time (Sec)	Connect Time (Sec)	CPU Time (Sec)	Connect Time (Sec)	CPU Time (Sec)	Connect Time (Sec)	CPU Time (Sec)
1	11. PCDB	5.07								
2	15. PCDB	5.20	15. PCDB	5.05						
3	21. PCDB	5.24	21. PCDB	5.26	21. PCDB	4.98				
4	24. PCDB	5.14	27. PCDB	5.18	29. PCDB	5.14	27. PCDB	5.04		
5	30. PCDB	5.21	33. PCDB	5.14	34. PCDB	4.99	31. PCDB	5.16	34. PCDB	5.24
6	32. PMS	18.73								
7	16. PCDB	5.19	38. PMS	18.88						
8	22. PCDB	5.11	23. PCDB	5.05	46. PMS	19.03				
9	28. PCDB	5.37	28. PCDB	5.19	29. PCDB	5.07	53. PMS	19.09		
10	28. PCDB	5.20	35. PCDB	5.14	33. PCDB	5.15	36. PCDB	5.28	62. PMS	18.93
11	23. PCDB	5.22	57. PMS	18.81	57. PMS	19.20				
12	27. PCDB	5.19	77. PMS	18.87	75. PMS	19.20	77. PMS	19.17		
13	26. PCDB	5.26	101. PMS	20.27	105. PMS	19.15	106. PMS	19.45	103. PMS	19.76
14	26. PCDB	5.11	29. PCDB	5.32	64. PMS	19.04	63. PMS	19.40		
15	26. PCDB	5.23	29. PCDB	5.15	34. PCDB	5.16	72. PMS	18.77	72. PMS	19.26
16	30. PCDB	5.31	34. PCDB	5.22	87. PMS	18.96	86. PMS	19.36	88. PMS	19.82

3.1.3 Contention Between Query and Load

In some applications the data base may be updated dynamically as users are accessing it. This can occur when the data is very volatile or when large volumes of data must be added in real time to keep up with the input rates. The latter of these is the case with the PMS application. Large volumes of data must be ingested in real time or the backlog would result in lost data. In order to see what impact may exist to the load rates if users were simultaneously querying the data base, a set of tests were devised which would demonstrate load rates with from zero to five contending users querying the data base. This test was performed at the request of the PMS project team and was done using only ORACLE Version 2.3.2.

ORACLE Version 2.3.2 Results

A PMS-like data base with 5,000 PMS headers was used as a base point to conduct the test. The HEADER table used had the following makeup:

HEADER Table

KEY*	SID	MID*	SSC	PLP	SDF	SHID	SIEC	TIME	PLS	SECHDR	UIC	COMMENT

* Indexed Field

The 5,000 row table was increased by 720 rows during each of six tests. In the first test, only the load was present, which serves as the control run. In the other five tests, a job was submitted from one to five times, respectively, which made 250 queries of the following nature:

SELECT • FROM HEADER WHERE KEY = ________

The KEY value used was different for each of the 250 queries but the same 250 values were used for each job submitted.

The table below shows some of the results obtained from these tests. These results were obtained from the VAX accounting file. It should be noted that a single query job without contention from a load job used about 78 seconds of connect time and 50 seconds of CPU time.

ORACLE QUERY AND LOAD CONTENTION RESULTS

Number of Query Jobs Present	Load Time for 720 Headers		Average Query Time Per Query Job	
	Connect Time (Sec)	CPU Time (Sec)	Connect Time (Sec)	CPU Time (Sec)
0	233.0	127.66	-	-
1	273.0	130.06	160.0	51.41
2	321.0	130.74	206.5	50.63
3	381.0	132.90	259.7	49.92
4	438.0	132.52	312.2	49.57
5	519.0	136.50	385.2	51.57

These results demonstrate a degradation in load performance that is proportional to the number of query jobs present. In the control run, a load rate average of 3.09 headers per second is obtained. With a single query job present the rate drops to 2.64 and with five query jobs present the rate falls to 1.39.

3.2 DBMS System and Computer System Options

The purpose of the tests defined under this section is to determine the impact of parameters that can be specified to constrain or enhance DBMS performance. The section is divided into two areas - one for parameters strictly selective under DBMS control, and the second which is related to parameters selective through the computer operating system under which the DBMS is to operate.

3.2.1 DBMS System Parameters

The flexibility offered by different DBMSs for the selection or definition of parameters that impact performance vary greatly from system to system. Some offer little if any "tuning" while others offer a wide variety of selective options such as variations in I/O buffering of information, temporary or internal work area sizes, the number of concurrently active data bases, the size of caches to reduce I/O, the use of backup/ recovery options, and the number of simultaneous users. Because these types of parameters are DBMS dependent, a direct comparison of the results of one DBMS to the results of another is meaningless and each section should be studied and assessed independently.

3.2.1.1 ORACLE DBMS System Parameters

In ORACLE Version 2.3.2, the flexibiity offered the data base user for varying DBMS parameters was limited. The revised Version 3.0 promised to depart significantly from that in Version 2. For these reasons, formal testing of DBMS system parameters under Version 2.3.2 was not performed.

However, under Version 3.0, several of the options available to the user were tested and the results of that testing will be presented here. Those options that were tested were 1) varying the number of cache buffers, 2) altering the space definition, 3) clustering data according to the value in a column or columns of a data base table, and 4) loading alternatives.

3.2.1.1.1 Buffering

A default system parameter file is included with the ORACLE 3.0 package. The file contains various parameters, such as maximum number of tables which may be defined, default data base file name, and the maximum number of concurrent ORACLE users. While most of the default values supplied were sufficient for our testing, it appeared that perhaps one parameter, the number of cache buffers, should be varied to determine its impact on performance. A cache buffer is an internal buffer area used by the data base management system that increases the likelihood that often used disk pages are available in main memory, eliminating the need to perform a disk I/O operation.

The default supplied with the ORACLE package is 50 buffers. In the testing, this value was increased to 100 and 200 buffers. A data base was first loaded with 5,000 PMS-like headers. The table structure employed in the testing was:

HEADER Table

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SDF	MESSAGE	HEADER

* Indexed Field

After each data base was loaded, two further tests were performed. In the first, 5%, or 250 records in the data base were accessed by issuing the following SQL command;

```
SELECT * FROM HEADER WHERE PRIMARY_KEY = ______
```

In the second test, the same query was performed, but there were 5 users querying the data base simultaneously instead of a single user.

All results appear in the tables below, with a brief interpretation of results following the final table.

ORACLE LOAD RESULTS

No. of Buffers	Average Insertion Rate (Hdrs/Sec)	% Improvement in Average Insertion Rate
50	5.33	---
100	6.72	+ 26.1
200	7.27	+ 36.4

ORACLE LOAD RESULTS

No. of Buffers	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
50	627.81	10,536	17,943
100	590.36	4,148	11,112
200	588.07	2,426	26,968

ORACLE QUERY RESULTS-SINGLE USER

No. of Buffers	Total Connect Time (Sec)	% Improvement in Total Connect Time	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
50	24.22	---	15.38	536	2,668
100	23.62	+ 2.5	16.05	475	3,599
200	22.25	+ 8.1	16.31	399	3,860

ORACLE QUERY RESULTS-5 USERS

No. of Buffers	Mean Total Connect Time (Sec)	% Improvement in Mean Total Connect Time	Mean Total CPU Time (Sec)	Mean Total Direct I/O's	Mean Total Page Faults
50	87.54	---	14.54	372	2,840
100	89.55	- 2.3	14.68	315	3,303
200	84.37	+ 3.6	14.32	173	3,924

The variation in the number of buffers had a significant impact on the load performance. One hundred appeared to be a good choice for the optimal value in this application. Whereas the results of the 200 buffer run were better than either the 50 or 100 buffer results, the improvement was not as great going from 100 to 200 buffers as going from 50 to 100 buffers. This can be seen easily in the second table. The improvement in CPU Time was only nominally better when 200 buffers were employed as opposed to 100 buffers. While the total number of direct I/O's continued to drop as the number of cache buffers increased, the number of page faults actually started to increase with 200 buffers. For this reason, and also because 200 buffers may be an unreasonable number to request in a non-standalone environment, it seemed more appropriate to request fewer. The results of querying, both for a single user and 5 users, showed that no significant variation appeared as a result of altering the number of cache buffers available.

3.2.1.1.2 Space Definition

At table creation time, an ORACLE user may choose to accept the default space allocation or may choose to alter the space definition to suit the particular table. If a priori information is known about the table, such as an approximation of the number of rows to be inserted into the table, then the user may better determine the space required for the table. An a priori knowledge of the fields to be indexed can also provide much assistance in determining useful space allocations. While the default specification may be acceptable for most applications, a test was conducted to determine what impact tuning of the space definition might have on load performance. A PMS-like data base was loaded twice, once using a variation of the default space definition, and once using a space definition which was based on 5,000 PMS headers being loaded, with indices on the PRIMARY_KEY, MID_SID, and TIME fields. The two space definitions are given below.

Initial space definition:	Initial Allocation - 25 pages	
	Increment - 25 pages	Data
	Max. No. of Increments - 10,000	Pages
	% Free on Each Page - 10	
	Initial Allocation - 25 pages	
	Increment - 25 pages	Index
	Max. No. of Increments - 10,000	Pages
Redefined space definition:	Initial Allocation - 1,600 pages	
	Increment - 300 pages	Data
	Max. No. of Increments - 6	Pages
	% Free on Each Page - 10	
	Initial Allocation - 600 pages	
	Increment - 100 pages	Index
	Max. No. of Increments - 6	Pages

A summary of the results of loading the data base using each of the space definitions defined above appears in the following tables.

ORACLE LOAD RESULTS

Space Definition	Average Insertion Rate (Hdrs/Sec)	% Improvement in Average Insertion Rate	Total Connect Time (Sec)
Initial	5.09	---	983.
Redefined	5.33	+ 4.7	938.

ORACLE LOAD RESULTS

Space Definition	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
Initial	667.40	10,898	26,596
Redefined	627.81	10,536	17,943

A 4.7% improvement in average insertion rate was realized by altering the space definition to the particular table. While this improvement is not dramatic, it serves to demonstrate that improvement may be observed by redefining and refining the space definition.

3.2.1.1.3 Clustering

One of the more important differences between ORACLE 3.0 and 2.3 was the addition of clustering in Version 3.0. Clustering can serve two purposes. When performed on more than one table, it can place records from the various tables physically close to one another according to the value in the "cluster key" field or fields. When table joins are performed on that key field, the data to be joined is theoretically on the same block, thus serving to hasten the join process.

When performed on one table, as in this testing, clustering can place records with the same value of the "cluster key" physically close to one another. If a query is performed on that field, all rows may be retrieved more quickly.

Clustered data exhibit certain characteristics - 1) each new value of the "cluster key" is placed on a new ORACLE disk block, 2) an index is created with each value of the "cluster key" stored once and with the index pointing to the first occurrence of the value of the "cluster key" and, 3) the clustered field (or fields) is stored once for the entire group of records with any one value.

Various tests were conducted to determine the affects of clustering and to compare clustering with indexing. Each test employed the PMS application with 5,000 header records.

In the first set of tests, a PMS-like data base was loaded four times. In the first, there were no indices and no clusters. In the second, there were no clusters but there was an index on MID_SID. In the third, there was a cluster on MID_SID but no indices, and in the last, there was an index on MID_SID and a cluster on MID_SID.

After each data base was loaded, the following query was performed 25 times (except for case 1 where no query was performed):

SELECT * FROM HEADER WHERE MID_SID = _____

Because there were 10 unique MID values and 3 unique SID values, or 30 unique combinations of MID and SID in the data base, each select solicited approximately 167 responses. The results of loading and querying are given in the summary tables below.

ORACLE LOAD RESULTS

Data Base Description	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
No Cluster, No Indices	18.59	---	269.	233.	19	3,956
No Cluster, Index MID_SID	11.55	+ 37.9	433.	360.	459	7,336
Cluster MID_SID, No Indices	3.10	+ 83.3	1,615.	759.	54,639	38,303
Cluster MID_SID, Index MID_SID	3.04	+ 83.6	1,644.	767.	54,605	38,174

ORACLE QUERY RESULTS

Data Base Description	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
No Cluster, Index MID_SID	6.8	---	170.	104.	3,960	7,535
Cluster MID_SID, No Indices	9.3	+ 36.7	232.	201.	571	7,403
Cluster MID_SID, Index MID_SID	9.6	+ 41.2	239.	205.	570	7,031

The addition of a cluster seemed to have a severe impact on performance, both in loading and querying. A close look at how ORACLE performs clustering lended some insight into this phenomenon.

By issuing a query on one of the ORACLE system tables, it was determined that each ORACLE block (1 ORACLE block = 2 disk blocks) held 7 header records. There were only 30 unique values for MID_SID, so approximately 24 ORACLE blocks were needed to hold the records associated with each unique "cluster key" value. Because the input data records to be loaded were ordered randomly according to the "cluster key" MID_SID, ORACLE necessarily performed an excessive number of block chains linking all of the blocks associated with any particular value of MID_SID. Each time a new record was added, the chain was traversed from the beginning to locate the first available space to add the record. This is evidenced by the dramatic increase in the number of direct I/O operations in the cluster loads.

The same type of behavior was exhibited in the query statistics. Because one of the basic tenets behind clustering is speedy retrieval, the results were particularly discouraging. However, this also can be explained for this case. The query which was executed chose all of the records with the selected value of MID_SID. It must be remembered that approximately 167 rows were retrieved in each select. Through discussions with ORACLE Corporation (ORACLE vendor), it was determined that the method currently being used in ORACLE retrieves the first row and notes that the first row was the last one retrieved. When ORACLE proceeds to select the next row, it checks to see which was the last one selected and knows to retrieve the next row. However, the entire chain from the beginning to the selected row must be traversed each time a new row is retrieved. If only a small number of rows are to be retrieved, the problem is not a serious one. However, in this case, where approximately 167 rows were retrieved for each of 25 queries, the problem surfaced as a serious one.

Two possible solutions to the problem were suggested by ORACLE Corporation personnel. First, it seemed important to select a "cluster key" so that the duplication factor of that key was smaller than in the first test. It is important, though, not to choose a key with too few duplications, because each new key is placed in a new block and a small duplication factor would result in much wasted space. In this case, for example, it would seem that since 7 headers can be placed in one block, a key which would cause far fewer than 7 headers to be placed in a block might be a bad choice.

A second possible solution suggested was to determine whether ordering the input data according to the "cluster key" MID_SID would have an impact on load performance.

The results of tests conducted using ORACLE Corporation's suggestions are shown below. In the first table, the results of three loads are shown. In the first, a cluster was created on the combination of the MID_SID field and the SDF field. There were 30 unique values for MID_SID and four unique values for SDF, or 120 unique values for the combination. In the second load, no cluster was created, but the same fields, MID_SID and SDF, were indexed. In the third run, the same two fields were clustered and in addition, indices were created for the PRIMARY_KEY and TIME fields.

ORACLE LOAD RESULTS

Data Base Description	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
Cluster MID_SID, SDF, No Indices	6.28	---	796.	436.	15,739	17,928
No Cluster, Indices on MID_SID, SDF	8.56	- 36.3	584.	485.	1,013	9,509
Cluster MID_SID, SDF, Indices on PRIMARY_KEY, TIME	3.17	+ 49.5	1,578.	813.	33,407	34,719

In the first run, there were approximately 42 rows for each unique combination of the "cluster key" MID_SID and SDF. With 7 records per block, approximately 6 blocks were needed to hold each "cluster key" combination.

The number of direct I/O's dropped significantly from 54,600 in the previous test to 15,700 in this test. Likewise, the CPU time dropped dramatically. The loading was still slower than loading with the indices created for the two fields, however. This should be expected though, because ORACLE must build an index for the "cluster key" in addition to clusteriing the data. The third test was run to determine the kind of results which could be expected in a more realistic situation where other indices, such as one on the PRIMARY_KEY, were necessary.

The query which was performed on the three data bases was:

```
SELECT * FROM HEADER WHERE MID_SID = _____
                           AND SDF = _____
```

The query was performed 25 times on each data base, and the results are given here.

ORACLE QUERY RESULTS

Data Base Description	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
Cluster MID_SID, SDF No Indices	1.4	---	35.	30.	172.	2,821.
No Cluster, Indices on MID_SID, SDF	3.5	+ 150.0	88.	53.	1.935.	3,241.
Cluster MID_SID, SDF, Indices on PRIMARY_KEY, TIME	1.4	---	35.	30.	171.	2,616.

When the cluster key was chosen as being comprised of two fields instead of one, as before, the average response time was much faster than by indexing. It should be noted that the results of this table should not be compared directly to the results of the previous query table. In that table, approximately 167 rows were retrieved in the average response time given. In this table, approximately 42 rows were retrieved in the average response time given. The proper adjustments must be made for direct comparison. The important point to be made is that clustering can outperform indexing under proper conditions, but can be a hindrance otherwise.

The purpose of the next test was to determine whether cluster load performance could be improved by ordering the input according to the value of the "cluster key." The table below shows the results of a load where the data was input randomly according to the "cluster key" MID_SID, and where the data was presorted by MID_SID in ascending order.

ORACLE LOAD RESULTS

Input Ordering	Average Insertion Rate (Hdrs/Sec)	% Improvement in Average Insertion Rate	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
Random	3.10	---	1,615.	759.	54,639.	38,303.
Ascending by MID_SID	11.31	+ 264.8	442.	390.	24.	5,656.

It can readily be seen that preordering of input data can have a significant affect on load rates. While this may not always be a viable option, the possibility of preordering should be considered.

It was of interest to determine the affect, if any, of clustering on the retrieval of records according to the value in the PRIMARY_KEY field.

The query which was performed was:

SELECT * FROM HEADER WHERE PRIMARY_KEY = ______

The query was performed on 5%, or 250 records in the data base.

Three scenarios were tested using this query. In the first, there was no clustering, only an index on the PRIMARY_KEY field. In the second run, there was a cluster on MID_SID and an index on the PRIMARY_KEY field. In the final run, there was a cluster on MID_SID and SDF as well as an index on the PRIMARY_KEY. The results are shown here.

ORACLE QUERY RESULTS

Data Base Description	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
No Cluster, Index on PRIMARY_KEY	.10	---	25.	16.	526.	3,212.
Cluster MID_SID, Index on PRIMARY_KEY	.35	+ 250.0	88.	40.	3,219.	4,788.
Cluster MID_SID, SDF, Index on PRIMARY_KEY	.16	+ 60.0	40.	22.	1,135.	3,847.

As can be seen here, clustering can have a significant impact on query performance when the query is on a field other than the one defined in the cluster, even if the field being queried on is indexed. Thru discussions with an ORACLE Corporation representative, it was determined that an index created on a field cannot be as efficient when the table that field resides in is clustered, even if the cluster is on another field, as when no cluster exists. While the index was still used in the above query to locate the proper rows, the index pointed to the first row in the cluster of rows where the desired row was located. The chain was then traversed until the proper row was located. The reason for this is that the clustered field (or fields) is stored only once (at the beginning of the cluster) for each unique value of that field (or fields). When a row is retrieved, that field (or fields) must be retrieved in addition to the remainder of the row. This is evident in the above table. The total number of direct I/O operations as well as the total CPU time in each run reflects the amount of chaining existent in the data base.

It is evident from the testing presented in 's section that clustering can significantly impact data base performance. It is evident also that caution must be used in defining a cluster so as to insure that the impact is not such as to create a degradation instead of improvement in performance.

3.2.1.1.4 Loading Alternatives (ODL)

ORACLE provides the user with a number of ways to load a data base. One method is by the execution of the ORACLE "INSERT" command embedded in a Host Language Interface (HLI) program. The FORTRAN program reads and processes one record at a time. A second method of loading is the ORACLE Data Loader, or ODL for short. In order to use the data loader, the input data file must be in raw data (binary) form.

This testing focused on the PCDB application which is described in Appendix II. Briefly, there were 5 tables - the TAPE table, FLE table (the term FLE was used because FILE is an ORACLE reserved word), ITEM table, CAT table, and ITEM_DESCR table. There were 13,631 input data records. The data base was loaded two times.

In the first load, data was inserted to the data base by use of the HLI "INSERT" command. In the second load, the ODL was used. The ODL program reads the user's input data, performs the mapping from raw data to a data base table, and then loads the data records into the data base table. In this testing, the data was preprocessed to get it into raw data form.

A summary of the results of the two methods of loading is given in the following table, as reported in the VAX accounting log.

ORACLE LOAD RESULTS

Load Description	Average Insertion Rate (Hdrs/Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
Use of "Insert"	4.68	2,914.	2,442.	12,225.	57,412.
ODL	4.46	3,059.	2,443.	19,421.	66,499.

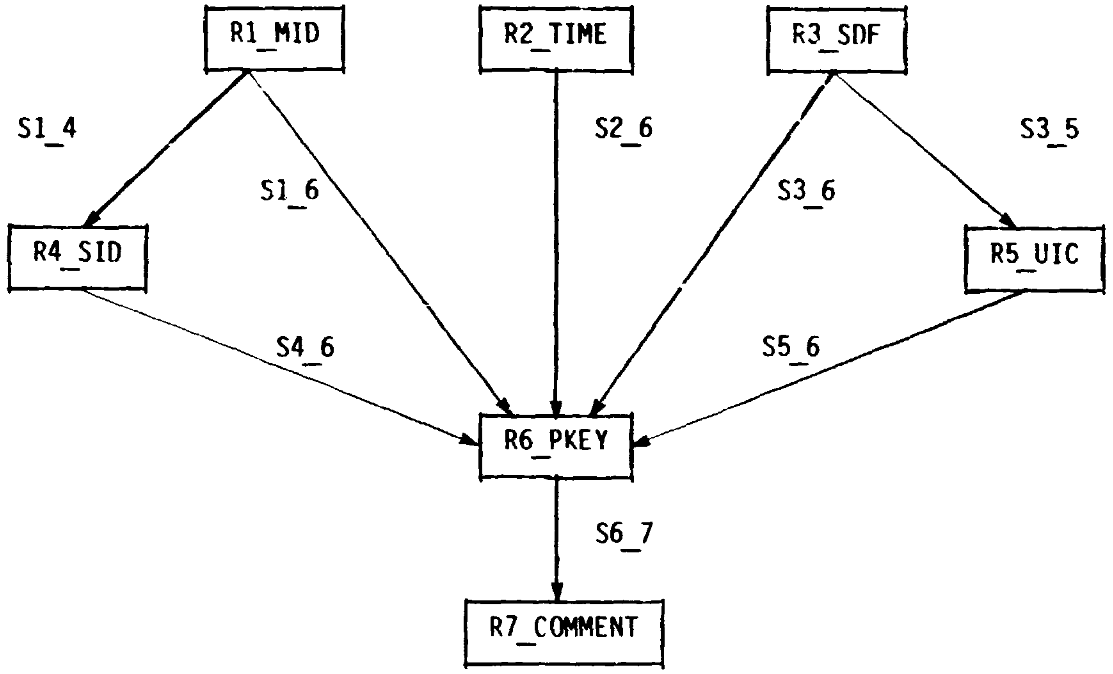
There was very little difference in our test between the load results of the two methods. If the input data is already in raw data form, the ODL load is probably an easier, faster method to use because no FORTRAN interface need be written. However, if the input data must be preprocessed in order to use the ODL loader, then perhaps the first method would be simpler. Another drawback to using the ODL loader is that the input data must be able to map directly into a column of a table. An example of where ODL could not be used is in the PMS application. The field MID_SID is a concatenation of the input data fields MID and SID.

While the CPU times of the two runs were almost identical, the larger number of direct I/O's and page faults contributed to the greater total connect time in the ODL load. The total time difference of 2.5 minutes out of 50 total minutes is still not very significant. Under different circumstances (more complex data base design or more records, for example), one method could prove to be more superior to the other in load rates.

3.2.1.2 SEED DBMS System Parameters

Tests were made on a SEED data base to determine the effect of three separate user controlled DBMS parameters on performance - journaling, an alternate hashing technique, and buffering. In all tests, the data base structure was the same as that described in previous sections and shown below. The tests were performed using SEED Version B.11.9.

SEED Schema



3.2.1.2.1 Journaling

Because in most real applications, some form of journaling is necessary, the first group of tests were made to determine the impact of journaling on SEED DBMS loading peformance. In all tests, 5,000 PMS-like header records were loaded.

Various types of journaling are available to SEED users. The first, TRNSCT, allows definition of a transaction containing one or more data base operations, which through use of this command, can be made permanent or nullified. The command takes the form of a FORTRAN subroutine call. Other journal modes which are available are: integrity (recovery but no roll forward or roll backward), roll forward, and roll backward. In integrity mode, a single data base update which did not terminate normally may be recovered. In roll forward, "after-images" are written to a file as updates are being performed, so that in the case of a system failure, the user may take a backup version of the data base and roll forward to a point prior to the crash which destroyed the data base. In roll backward mode, while the user may still roll the data base forward as described above, "before images" are written to a file, so that the data base may also be rolled back to a previous state. This is particularly useful to correct user errors in the data base.

To get a sampling of the effect of various modes of journaling on load rates, tests were made comparing the results of no journaling, TRNSCT, and roll backward. These three modes were chosen because they represent no journaling capability, a simple type of journaling, and the most complex form of journaling which SEED provides. The results are shown in the table below.

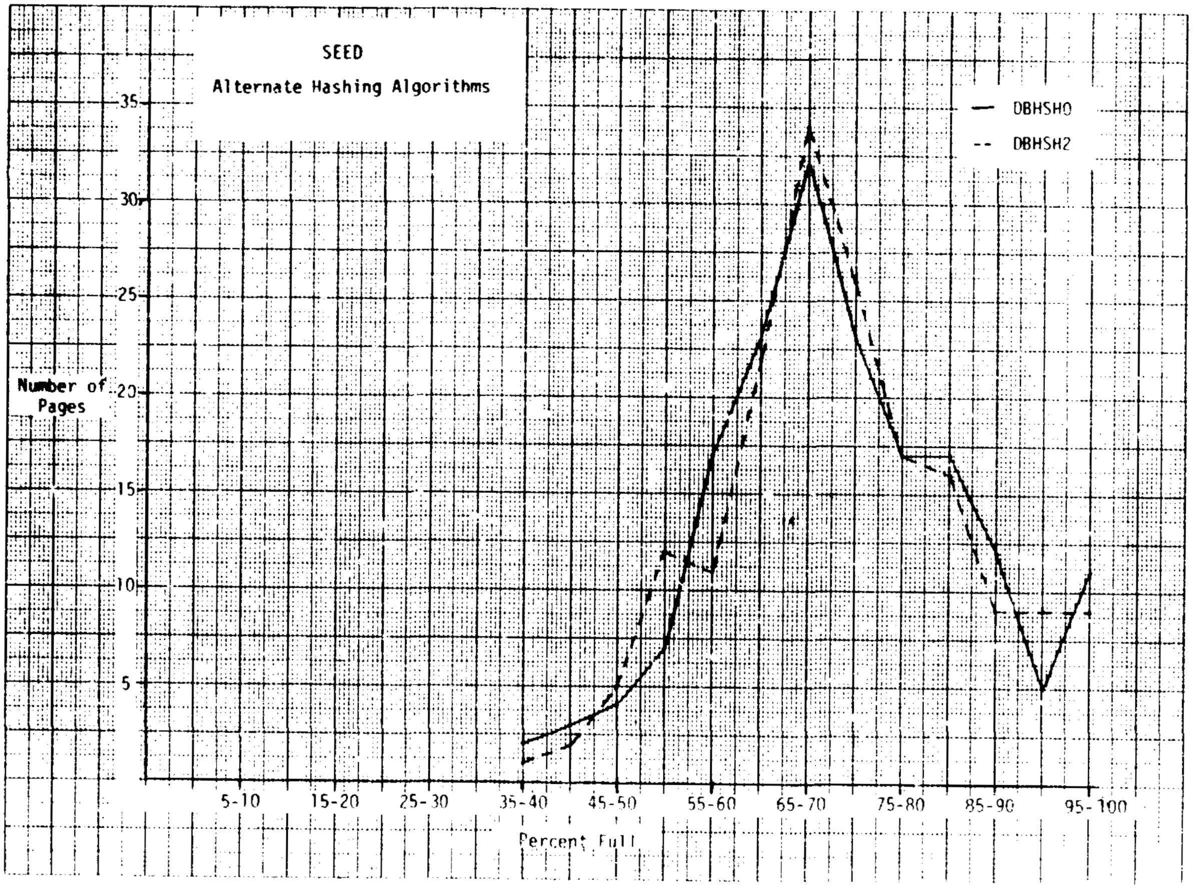
SEED LOAD RESULTS

Journal Mode	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
None	6.2	---
TRNSCT after every record	6.1	+ 1.6
TRNSCT after every burst	6.2	---
Roll Backward	2.5	+ 59.7

Whereas the use of TRNSCT does not seem to have much affect on load rates, the use of journal mode roll backward severely impacts performance.

3.2.1.2.2 Alternate Hashing Technique

The user has a choice of hashing algorithms to use to optimize placement of records in the data base according to the value of a "key" variable. The SEED DBMS supplies two such algorithms and also gives the user the option of supplying up to 7 other algorithms. The first SEED supplied algorithm (DBHSH0) should optimize placement of "integer" type key variables (or any variable ≤ 4 bytes long), while the second algorithm (DBHSH2) should optimize placement of "character" type key variables longer than 4 bytes. In the PMS application, the key variable PKEY, is a 10 byte character string. In this group of tests, a data base was loaded two times, once using DBHSHO and once using DBHSH2. The graph below shows that, for this application, there did not seem to be a significant difference between DBHSHO and DBHSH2 in the placement of data within the data base.



3.2.1.2.3 Buffering

One of the features that the SEED DBMS employs is the ability of the user to manipulate buffering at run time. The purpose of the group of tests in this section was to evaluate load performance under three different buffering strategies. In the first strategy, 50 buffers were requested. All 50 buffers were to be left unwritten (this gains efficiency but exposes the program to system crashes). All buffers operated in "least recently used" mode as opposed to "adaptive" mode. In LRU (least recently used) buffering, the page that was least recently accessed is replaced by the new page in core. In adaptive buffering, a count is kept of the number of times a page has been accessed, and the page with the least number of accesses is replaced.

In the second buffering strategy, 100 buffers were requested, all to be left unwritten, and all to act in LRU mode.

In the final buffering strategy, 50 buffers were requested, all to be left unwritten, but 39 to act in LRU mode and 11 to act in adaptive mode. This hybrid loosely results in the least accessed adaptive buffers being used to hold the least recently used LRU buffer into which the new page will be read.

A comparison of the results of the three buffering strategies is shown below.

SEED LOAD RESULTS

Buffer Strategy (number of buffers, number to be left unwritten, number in LRU mode)	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate
(50,50,50)	6.2	---
(100,100,100)	8.3	- 33.9
(50,50,39)	4.9	+ 21.0

While the results for the second strategy (100,100,100) are far better than the others, in a realistic atmosphere (i.e. non-standalone), it would probably be unreasonable to request such a large number of buffers. The default number of buffers is 4. A buffer is defined as the size of the largest page in the data base.

The primary conclusion to be drawn from this study is that buffer manipulation can have a significant impact on performance and by altering the parameters, an optimal scenario can be reached for a particular application.

3.2.1.3 INGRES DBMS System Parameters

The tests which were performed using the INGRES DBMS were designed to measure the effect of secondary indices, storage structure, journaling, and loading alternatives on the performance of the DBMS. In all testing, the PCDB application was used. This application is described in detail in Appendix II. The various methods of storing data are discussed in Section 3.2.1.3.2.

3.2.1.3.1 Secondary Indices

The primary key is the column, or columns, of the table which determine where the rows of the table are stored. For example, if the storage structure of the table is heap, there is no primary key. Each row is stored in the data base in the next available location. There is no ordering of rows. When the storage structure is defined as hash, a primary key must be defined. It is the value in this column, or columns, which determines the storage location. In an "ISAM" structured table, the rows are actually stored in sorted order according to the values in the primary key column (or columns).

Because queries are performed which cannot take advantage of the storage structure of the table (i.e. the where clause contains a field not in the primary key), INGRES offers another solution. By creating secondary indices on fields other than the primary key fields, query optimization can be enhanced.

The purpose of the tests in this section was to measure the impact of the presence or lack of secondary indices on query response time. These tests were run under INGRES Version 1.3. In each test, the query that was executed was:

```
RANGE OF F IS FILE, C IS CAT
RETRIEVE (C.CAT, C.CATEGORY, C.FUNCTION, F.FILENUM, F.FLSTART,
    F.FLSTOP, F.FLFIRSTORB, F.FLLASTORB, F.FLLEN) WHERE
F.TAPEID=C.TAPEID AND
(C.FILENUM=F.FILENUM OR C.FILENUM= 0) AND
F.NUMITEMS=0 AND
C.CAT = 'OZONE' AND
(F.FLSTART<='710401000000' AND F.FLSTOP>='700801000000') AND
F.TAPEID='DPFL*'
```

Each time the query was performed on the data base, 2,796 rows were retrieved out of a total of 13,651 records in the entire data base.

The table structures were defined as follows:

Table TAPE was 'hash'ed on TAPEID

Table FILE was 'ISAM'ed on TAPEID, FILENUM

Table ITEM was 'hash'ed on TAPEID, FILENUM, ITEM

Table CAT was 'hash'ed on TAPEID, FILENUM, ITEM, CAT

Table ITEM_DESCR was 'hash'ed on ITEM

The table below shows the results of the tests where various secondary indices were created. For each run, the field(s) with a secondary index is shown, along with the index type (hash or ISAM) for that field. The statistics were obtained from the VAX account file.

INGRES QUERY RESULTS

Job	Secondary Index Field	Index Structure	Average Response Time (Sec)	% Improvement in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's
1	None	---	.053	---	148.	92.01	2,189.
2	CAT	Hash	.055	- 3.8	153.	91.79	2,197.
3	FLSTART	ISAM	.054	- 1.9	151.	91.70	2,195.
4	FLSTOP	ISAM	.054	- 1.9	150.	92.56	2,195.
5	CAT, FLSTART	Hash,ISAM	.053	---	149.	92.09	2,199.
6	CAT, FLSTOP	Hash,ISAM	.054	- 1.9	150.	92.35	2,199.
7	FLSTART, FLSTOP	ISAM,ISAM	.054	- 1.9	150.	92.02	2,195.
8	CAT,FLSTART, FLSTOP	Hash,ISAM, ISAM	.054	- 1.9	150.	92.30	2,200.

There is a 4% maximum difference in average response time and less than 1% difference in total CPU time.

The query involved only the FILE and CAT tables. The FILE table consisted of 13,501 rows and the CAT table consisted of 55 rows. Two explanations may be given as to why the presence of secondary indices had little effect on the query rates. First, the query joined the CAT and FILE table on TAPEID. Because the CAT table was small, and the value 'OZONE' was specified in the where clause of the query, the INGRES DBMS could take advantage of the size of the table and quickly locate the rows in the CAT table which satisfied the query. Because the FILE table was 'ISAM'ed on TAPEID and FILENUM, the files for any particular TAPEID were arranged in ascending order by TAPEID and, therefore, only a small part of the FILE table was read for each join with a TAPEID in the CAT table.

Second, a secondary index is generally more efficient if the total number of rows retrieved by the query is smaller than the number of pages in the primary table. The query performed in this testing retrieved 2,796 rows while the FILE table occupied only 1,126 pages, of which only a small number were read (because of the ISAM structure of the FILE table).

While the variation in total number of direct I/O operations is small, there were fewer for the run where no secondary indices were present than for the others.

3.2.1.3.2 HEAP vs. HASH vs. ISAM

The three basic modes of storage available to the INGRES user are heap, hash, and ISAM. In heap structure, which is the default storage structure, there is no structure to the data. New rows are placed at the end of the table, so the order is random. Any time a query is executed against a table, the entire table must be searched. Hash structure stores each row at an address determined by the value in a column, or columns, of the row. When a new row is added to a table, INGRES calculates its address based on the value in this key column, or columns. Queries involving exact matches on the key field are greatly accelerated using the hash structure. ISAM (or Indexed Sequential Access Method) arranges rows in ascending order of key value and then subdivides them into pages. Then the largest key value on each page is collected, and these are sorted in a tree structured index. Access to an ISAM table is achieved by searching the index for the correct page and then accessing the page. Retrieval involving a range of values is particularly efficient using the ISAM method, as are exact matches.

A test was run to measure the effect of altering table structure on query response times. The runs in this test performed the same query that was described in the previous section and were run under INGRES Version 1.3. There were no secondary indices present in these runs. In the first run, both the FILE table and CAT table were 'heap' structure, that is, no structure at all. In the second run, both the FILE table and CAT table were 'hash' structure - the FILE table being 'hash'ed on TAPEID and FILENUM and the CAT table being 'hash'ed on TAPEID, FILENUM, ITEM, and CAT. In the third run, both tables were 'ISAM' structure, and in the last run, the FILE table was 'ISAM' structure and the CAT table was 'hash' structure.

The results of querying the data base for each of these situations is shown below, as obtained from the VAX accounting log.

INGRES QUERY RESULTS

Table Structure	Average Response Time (Sec)	% Improvement in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's
FILE*, CAT† - heap	.064	---	178.	102.14	3,141.
FILE*, CAT† - hash	.079	- 23.4	221.	111.78	4,721.
FILE*, CAT† - ISAM	.053	+ 17.2	149.	92.92	2,182.
FILE* - ISAM CAT† - hash	.053	+ 17.2	148.	92.01	2,189.

* FILE - primary key on concatenation of TAPEID and FILENUM

† CAT - primary key on concatenation of TAPEID, FILENUM, ITEM and CAT

Because the CAT table had only 55 records, the variation in structure of that table did not contribute significantly to the large variation in total connect time for the 4 queries. The structure of the FILE table, however, was responsible for these large variations. When the FILE table was 'hash'ed, for each of the 2,796 rows which were retrieved, the value of the hash key was determined and the row located. The total number of direct I/O operations of 4,721 reflects the fact that the FILE rows for each TAPEID were spread throughout the data base. When the FILE table was declared to be 'ISAM' structure, the FILE rows for each tape were located physically close to one another in ascending order of FILENUM. For each TAPEID, far fewer direct I/O operations were necessary to retrieve the 2,796 rows that satisfied the query. The faster retrieval was due not only to the ordering of FILE rows, but also to the fact that the entire table did not have to be searched, but only a small portion. The results of the query where no structure was placed on the FILE table show that this was still faster than 'hash'ing. This was attributable to the FILE table occupying 1,126 pages and the query retrieving 2,796 rows. There were less direct I/O operations than when the FILE table was 'hash'ed, showing further that when the FILE table was hashed, the records for each TAPEID were spread throughout the data base.

Because of the structure of the data being loaded into the data base, the FILE records for each TAPEID were read and loaded sequentially, even though the FILE table was declared 'heap'. The difference between this and the run where FILE was declared 'ISAM', was that when the table is 'heap' the entire table must be searched whereas when the table is 'ISAM', only part of the table is searched.

3.2.1.3.3 Journaling

In the final test implemented to measure the affect of INGRES DBMS parameters on performance, a PCDB data base (see Appendix 1-1) was loaded twice - once with no journaling and once with the journaling capability enabled. This test was conducted using INGRES Version 1.4.

The results of both runs are shown in the following table, with results from the VAX accounting log. In each run, the data base was loaded into 'heap' tables using the copy command.

INGRES LOAD RESULTS

Description	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
No Journaling	31.9	---	427.	327.35	3,121.	1,451.	570.
Journaling	26.4	+ 17.2	517.	360.32	3,123.	15,114.	576.

When the journal mode is enabled, a copy is kept of every transaction on the data base, so that the data base may be recovered after a failure or so that an audit may be made of transactions to the data base. While there was a 17% reduction in load performance when journaling was enabled, in a practical situation, journaling is probably necessary to insure the integrity of the data base.

3.2.1.3.4 Loading Alternatives

The INGRES DBMS currently offers three methods of loading a data base. Each method is discussed briefly here.

The "append" command loads data into the data base one record at a time or copies data from one table to another. By adding the word "repeat" to the front of the command, the append statement is compiled only the first time it is executed, and then saved for subsequent executions. The second method of loading a data base is by using the copy command, which copies data from a VAX VMS file into a table. With both of these methods, it is best to load the data without any structure, and then modify the structure appropriately. In the final method of loading, the data is loaded to a temporary table (without structure) using the copy command, and then this table is "repeat append"ed to the permanent table (with structure).

Using INGRES Version 1.3, several tests were conducted which attempted to compare the various methods of loading. In each test, a PCDB like application was adopted. In the PCDB application, five tables were present - a tape table, a file table, an item table, a cat table and an item description table. (This application is further explained in Appendix II). A total of 13,631 inserts were made into the data base in the first test. The test was run to compare the two methods of loading, copy and repeat append, on a heap structured data base. The results of the loads are presented below as reported in the VAX accounting file.

INGRES LOAD RESULTS

Type of Load	Average Insertion Rate (Hdrs/Sec)	% Degradation over "Copy"	Total Connect Time (Sec)	Total CPU Time (Sec)
Copy	31.6	---	431.	324.
Repeat Append	3.3	+ 89.6	4,105.	1,716.

There was a large variation between the two runs in total number of direct I/O's, buffered I/O's and page faults. These statistics are given below (from the VAX accounting log).

INGRES LOAD RESULTS

Type of Load	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
Copy	3,122	1,451	514
Repeat Append	30,511	97,116	4,340

While the use of copy may not always be a viable approach for the user, its use is far superior when loading large amounts of data into a data base.

The second test was designed to compare loading a data base where all five tables have been hash structured using the copy command, against loading temporary tables with no structure (heap) and then appending the temporary tables to permanent tables defined as hash structure (See Appendix II). This test was conducted using INGRES Version 1.3. In this test, 2,000 records were loaded. Again, the statistics reported were obtained from the VAX accounting file.

INGRES LOAD RESULTS

Type of Load	Average Insertion Rate (Hdrs/Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)
Copy into hash structured tables	3.6	549.	193.
Copy into heap tables		69.	51.
Append to hash structured tables		302.	105.
------------------		-----	----
Total	5.4	371.	156.

While the CPU time difference is minimal between the two methods (less than 40 seconds) the total connect time difference is somewhat larger (178 sec.). A look at the total number of direct I/O's is helpful (VAX accounting file statistic).

INGRES LOAD RESULTS

Type of Load	Total Direct I/O's
Copy into hash structured tables	29,143.
Copy into heap tables	521.
Append to hash structured tables	15,305.
-------------------	-------
Total	15,826.

There are close to twice as many direct I/O's on the run where the data is being copied directly into a hash structured table.

3.2.2 Computer Operating System Parameters

One of the computer system manager's responsibilities is to set up "profiles" on system parameters that affect all users and "profiles" for individual users which may vary according to a particular user's knowledge of the system and needs on that system.

One aspect of the computer operating system which is important to understanding and evaluating the results of all testing is test repeatability. It is important to be able to repeat any test under conditions that are as similar as possible to the original test. To this end, a PMS-like data base was loaded several times (5,000 records each time), with the operating system being "re-booted" between each run.

Another such parameter in a "profile" may be the working set size (the maximum number of pages a user's process may have). In an effort to erase any effect a large number of page faults might have on DBMS performance, the default working set size was increased in this testing, from 512 to 1,500 pages. The same PMS-like application was used in this testing to load a data base with 5,000 header records.

Finally, tests were made to determine if disk allocation of data base related software and files would seriously impact performance. The data base files, loading software, data to be loaded, and data base management system software were assigned in varying configurations on the available disk units. Tests were made which loaded the same data into an identically defined PMS data base each time but where the location of these four important components was reconfigured.

3.2.2.1 Impact of VAX/VMS Behavior on DBMS Performance

ORACLE Version 2.3.2 Results

In an effort to determine what VAX/VMS system variability contributes to DBMS performance, an examination of results obtained in other testing was made. In that testing, one or more repetitions of the same test had been performed under circumstances which would appear to be identical. The same ORACLE system was used, the same HLI FORTRAN software was executed, the same input data was used, and all tests were conducted in standalone mode. The data bases had all been reinitialized prior to starting the tests. The major unknown and uncontrolled variable in the tests was the state of the VAX/VMS system software and related page maps and buffers.

The structure of the HEADER table is shown here:

HEADER Table

KEY*	SID	MID*	SSC	PLP	SDF	SHID	STEC	TIME	PLS	SECHDR	UIC	COMMENT

* Indexed Field

A comparison of the results of six pairs of otherwise identical runs demonstrate that variability in DBMS performance does exist that can be attributed to the VAX/VMS system independent of the DBMS. Below is a table summarizing the percent difference in each of the six pairs of "identical" tests for connect time, CPU time, direct I/O operations, and page faults. The percentages in the table reflect the relative difference in performance within an identical pair of runs.

ORACLE PERCENTAGE VARIATION IN LOAD RESULTS

Pair	Process	Percentage Variation			
		Connect Time	CPU Time	Direct I/O's	Page Faults
1	Host Process	8	3	0	6
	Detached Process	8	0	0	0
2	Host Process	8	6	0	0
	Detached Process	8	1	0	0
3	Host Process	1	2	0	9
	Detached Process	1	0	0	0
4	Host Process	1	1	0	4
	Detached Process	2	1	0	0
5	Host Process	3	3	0	0
	Detached Process	2	1	0	2
6	Host Process	2	1	0	11
	Detached Process	2	2	0	23

The only constant between the pairs appears to be the direct I/O statistic. Connect time for both the FORTRAN software interfacing with the DBMS (Host Process) and the ORACLE detached process varies from one to eight percent. In only two cases was the CPU time constant. The only aspect of the DBMS that could contribute to the observed variability is the status of the data base files after reinitialization. These files may be in different states which would require different processing during the tests. The magnitude of this type of variation would probably be small so the primary explanation is attributed to the inherent variability present in a virtual memory operating system.

A difference in performance was noted between the loading of a data base which had been freshly created versus the same one after it had been reinitialized and reloaded with identical data. Below, the results of two pairs of tests are summarized in the same format as the previous table. Within each pair, the only difference in the test setup was that in one the data base was created new prior to loading while in the second the data base was reinitialized. In both pairs a significant increase in connect time was observed in the reinitialized data base load. The difference was 15% in both cases, which is nearly twice the maximum difference in connect time observed in the previous table. This indicates that the observed difference is attributable to more than simply system variation.

ORACLE PERCENTAGE VARIATION IN LOAD RESULTS

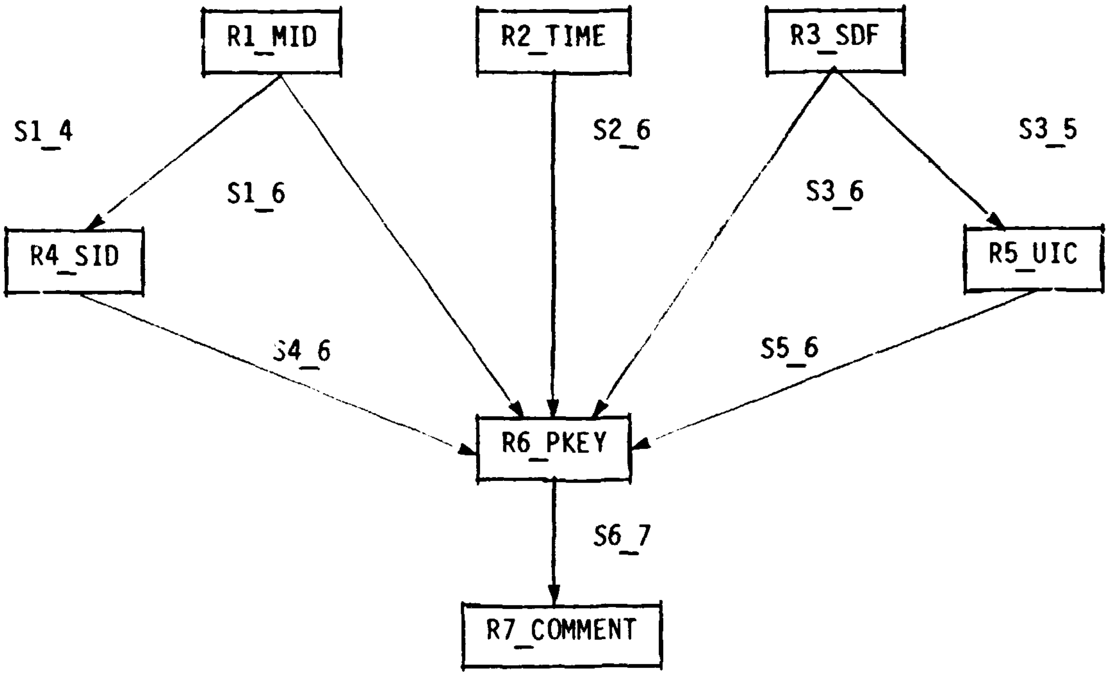
Pair	Process	Percentage Variation			
		Connect Time	CPU Time	Direct I/O	Page Faults
1	Host Process	15	3	0	13
	Detached Process	15	0	1	0
2	Host Process	15	2	0	6
	Detached Process	15	1	0	0

Apparently, after reinitialization the DBMS is less efficient than after the data base is created fresh. These results suggest that some fragmentation of space or similar degradation occurs when the data base is reinitialized, thus adding some overhead to the loading process when repeated.

SEED Version B.11.9 Results

The testing in this section was performed on the same data base schema that has been described in previous sections and is shown again here. In all tests, 5,000 PMS-like headers were loaded.

SEED Schema



A group of tests were made to show the variability that can occur in load rates when the same data base is loaded more than one time. In these tests, the data base was redefined for each run, and the computer was "re-booted" for each run. The results of loading the same data base four times are shown below.

SEED LOAD RESULTS

Job Number	Average Insertion Rate (Hdrs/Sec)	% Degradation in Average Insertion Rate	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
1	6.17	---	810.	435.	12,643.	15,086.
2	6.42	- 4.1	779.	437.	12,645.	15,882.
3	6.40	- 3.7	781.	439.	12,647.	15,306.
4	6.28	- 1.8	796.	436.	12,645.	15,158.

There is about a 2-4% difference in average insertion rate and less than 1% difference in total CPU time. While the number of direct I/O's was almost identical for all runs, the number of page faults was varied.

3.2.2.2 Working Set Size
ORACLE Version 3.0 Results

A test was performed to attempt to assess the impact of working set size on performance of a data base load. The working set is defined as the set of pages in memory to which a process can refer without incurring a page fault. In this test, 5,000 PMS-like headers were loaded into a data base two times. In the first run, the working set size was 512 pages and in the second, the working set size was 1,500 pages. The PMS header table is shown here.

HEADER Table

PRIMARY_KEY*	MID_SID*	TIME*	UIC	SOF	MESSAGE	HEADER

* Indexed Field

The results of the 2 loads are given in the following table.

ORACLE LOAD RESULTS

Working Set Size (Pages)	Average Insertion Rate (Hdrs/Sec)	% Improvement in Average Insertion Rate
512	5.20	---
1,500	5.23	+ .6

As can be seen in the results, the increase in working set size did not impact the performance of the load significantly. A glance at the peak working set size revealed that when the maximum working set size was 512, the peak working set size was 512. In other words, the process needed all 512 pages. When the maximum working set size was increased to 1,500 pages, the peak working set size increased to only 549 pages. Even though a large number of pages were available, the process only needed 549 pages. Therefore, no drastic improvement in performance was seen. One explanation for this might be that because ORACLE creates detached processes, the pages needed are divided up among the processes, so that even though the total may be large, the number of pages needed by each detached process is relatively small.

SEED Version B.11.9 Results

In an effort to assess the impact of working set size on performance and also to try to minimize the variance in number of page faults between runs, the working set size was increased from 512 to 1,500 pages. The schema that was used was identical to that described in Section 3.2.2.1. The results of loading two PMS-like data bases under these conditions (in addition to the conditions stated for the previous testing) are shown below. These tests were run in a standalone environment. These results can be compared directly to the results in Section 3.2.2.1, where 4 jobs were run to load the same data base. However, in those runs, the working set size was 512 pages. The results of those four runs have been averaged and also appear in the table below.

SEED LOAD RESULTS

Working Set Size (Pages)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
512	792.	437.	12,645.	15,358.
1,500	757.	423.	12,652.	1.
1,500	764.	427.	12,644.	1.

By increasing the working set size, the number of page faults stabilized at a very low number. The affect this had on total connect time was to lower it by about 4%. The variation in total connect (and total CPU) time between runs was lowered to about 1%.

These results show that while the large working set size used in this testing may be impractical under normal conditions, it can have a significant impact on data base performance. They also serve to demonstrate the variability inherent in the DEC virtual operating system.

3.2.2.3 Disk Allocation
ORACLE Version 2.3.2 Results

A variety of components (both hardware and software in nature) are present when the ORACLE DBMS is in use. The amount of control over these components and how they are configured certainly varies depending on the host computer's intended usage and the degree of importance of the data base application. The configuration of the software components and the data they access may be more under the control of the DBA or the data base user community. To ascertain what, if any, variability in DBMS performance could be found due to configuration, a set of tests were made which confined their investigation to the software components and data accessed.

At the time these tests were devised and conducted, the hardware configuration included two DEC RP06 (176 megabyte) disk drives which were configured on the same mass bus adapter. The components relevant to the testing included the VMS system software, the ORACLE "kernel" software, the FORTRAN load routine, the ORACLE data base files, and the input data file (PMS-like data to be loaded into the data base). Of these, the system software and data base files could not be varied. The system software was fixed on the system disk, designated DBA0, while the data base files had to reside on DBA1 because of space limitations on DBA0. The remaining three components (ORACLE "kernel" software, FORTRAN load software, and the input data file) were configured in various combinations after which a load of 5,000 headers into a PMS-like data base would be performed using the following table makeup:

HEADER Table

KEY*	SID	MID*	SSC	PLP	SDF	SHID	SIEC	TIME	PLS	SECHDR	UIC	COMMENT

* Indexed Field

The loads were all performed in standalone mode to eliminate the uncontrolled influence of other users. The data base was always reinitialized before the loads were made. The table below summarizes the results of each load. The FORTRAN load routine and the ORACLE detached process have been reported separately for closer scrutiny. Statistics include total connect time, CPU time, direct I/O operations and page faults as shown in the VAX account log. Probably the most relevant statistic is the total connect time since it will reflect the relative efficiencies or deficiencies observed in each configuration. Most of the configurations were tested a second time to see how much variability would exist.

ORACLE LOAD RESULTS

Component Placement	Job No.	Higher Total Connect Time (Sec)	Sum Total CPU Time (Sec)	Sum Total Direct I/O's	Sum Total Page Faults	% Degradation over Best Results
ORACLE Kernel s/w-DBA0 Input Data File-DBA1 FORTRAN Load s/w-DBA1	1	1,388.	799.20	8,426	6,894	---
	2	1,418.	816.16	8,454	6,493	+ 2.2
-DBA0 -DBA0 -DBA0	1	1,407.	808.13	8,426	7,268	+ 1.4
	2	1,421.	807.89	8,426	7,861	+ 2.4
-DBA0 -DBA0 -DBA1	1	1,450.	812.19	8,405	7,557	+ 4.5
-DBA1 -DBA0 -DBA1	1	1,557.	808.13	8,405	6,947	+ 12.2
-DBA1 -DBA1 -DBA1	1	1,562.	815.96	8,426	6,389	+ 12.5
	2	1,541.	808.88	8,438	6,190	+ 11.0
-DBA0 -DBA1 -DBA0	1	1,746.	821.10	8,404	6,740	+ 25.8
	2	1,618.	809.59	8,403	6,722	+ 16.6

The results display a great deal of variability even between repetitions of the same configuration. This variability is attributed primarily to the VAX/VMS operating system. Because of this, it is more difficult to draw definitive conclusions from the observed results. The most efficient configuration observed in these tests appears to be the first one in the above table where the input data file and FORTRAN load software were located on DBA1 and the ORACLE Kernel software was on DBA0.

This is followed closely by a configuration with all three components on DBA0 and then by a configuration with the FORTRAN software on DBA1 and the other two components on DBA0. The worst results were apparent in the configuration with the input data file on DBA1 and the other two components on DBA0. There is a 25% increase in connect time between the best observed results and the worst. Why such a difference exists is not readily apparent. The only difference between the best and worst case configurations is in the FORTRAN load software, where it resides on DBA1 in the best case and on DBA0 in the worst. This does not seem to warrant such a discrepancy in performance. Further analysis of the results shows slightly higher CPU times in the worst case and similar direct I/O's and page faults.

The variability between tests with similar configurations indicates that the impact of the VAX/VMS system software is significant and taints the conclusions that can be made between results of different configurations. It is worthy to note that the two disk drives were on the same mass bus adapter and must share the same communication "pipeline". Additional testing in the area of component location may be desirable when more drives are available and/or when multiple mass bus adapters are present.

SEED Version B.11.9 Results

In the testing performed in this section, the placement of various components of the load process was altered in an effort to assess the impact of actual position within the system on DBMS performance. In this testing, the schema was identical to that described previously and in Appendix I. Two RP06 disk drives were available - DBA0 and DBA1. They were both configured on the same mass bus adapter. In all testing, the actual data base and DBMS software resided on DBA1. The placement of the loading software and data to be loaded was varied. Each test was repeated at least one time. All results are shown below, with results obtained from the VAX accounting file.

SEED LOAD RESULTS

Component Placement	Job No.	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
Input Data File - DBA1	1	812.	452.	12,889.	19,995.
	2	833.	450.	12,889.	20,110.
FORTRAN Load s/w - DBA0	3	846.	453.	12,889.	19,977.
Input Data File - DBA0	1	809.	450.	12,890.	19,723.
	2	823.	460.	12,889.	19,258.
FORTRAN Load s/w - DBA0					
Input Data File - DBA1	1	815.	454.	12,889.	19,472.
	2	804.	451.	12,889.	19,662.
FORTRAN Load s/w - DBA1					
Input Data File - DBA0	1	820.	451.	12,889.	19,678.
	2	810.	450.	12,889.	19,375.
FORTRAN Load s/w - DBA1					

The maximum difference in total CPU time between all runs is 10 seconds or about 2%. The maximum difference in total connect time is about 5%. It appears that when only two disks are available and both are on the same mass bus adapter, the placement of the various load components on those disks should not be a serious consideration. Perhaps if more disk drives were available, placement would be more important and have a more significant impact on performance. This might be especially true if different areas of the SEED data base could be placed on different disks. It is also worth noting that the VAX system software was resident on disk DBA0 during these tests and may contend with the DBMS processing for I/O to the device.

4.0 SUPPLEMENTAL TESTING

In addition to the testing described in Sections 2 and 3, additional variations were made on the basic data base designs. Many of these variations were chosen to provide feedback to the PMS project team for pursuing their design. As different DBMSs offer different alternatives to the user, a direct comparison between tests run under one DBMS and another may not always be applicable. Any testing described in this section which is related to that done in either Section 2 or 3 will refer back to the appropriate section for cross reference. All testing performed in this section was conducted in a standalone environment.

In addition to the PMS-like application being used as a testbed in this and other sections of the report, testing in this section also includes the use of the PCDB application which is described in Appendix II. Some of the differences between the applications which contributed to the testing of both were 1) data base size, 2) complexity of design, and 3) major requirement (project goal) of the data base.

For most of the testing of the PMS-like application, the data base contained 5,000 records. The data base design was relatively simple and the main goal was to load 7 records per second.

On the other hand, the PCDB data base design was more complex and the data base contained 13,631 records. The main requirements of the PCDB data base was acceptable query response times.

4.1 Alternative Data Base Designs

The impact on performance of alternative data base designs is of significance to all DBMS users. This section provides additional information to that included earlier in Section 2. The results presented in this section lend some insight to how data base design can be "tuned" to optimize performance for a particular data base application.

4.1.1 ORACLE Version 2.3.1

The goal of some initial testing done with ORACLE was to determine how a PMS data base could be designed that met both the data management needs as well as the desired performance goals.

4.1.1.1 Load Performance - Prototype Design

A prototype design was evolved by the PMS project team. It is composed of a single table described in Appendix I and is pictorially summarized below:

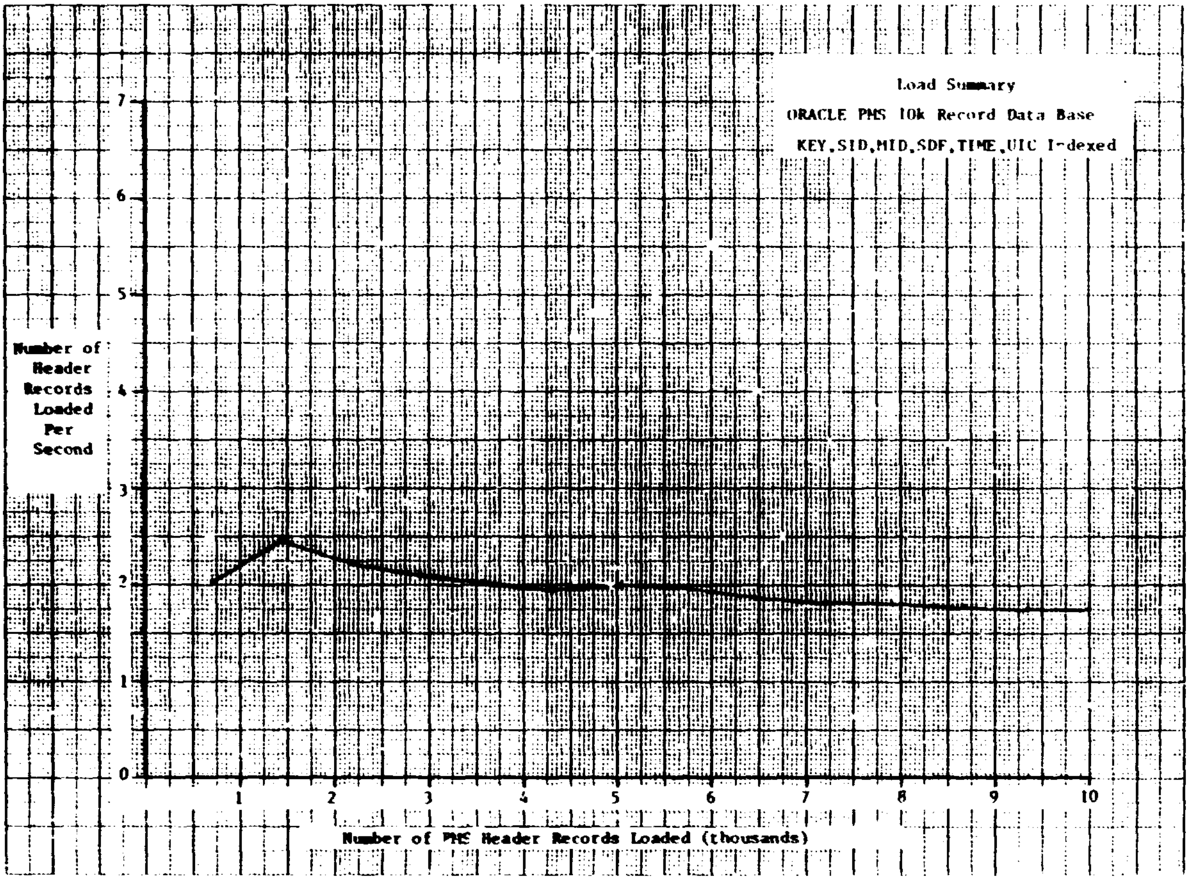
HEADER Table

KEY*	SID*	MID*	SSC	PLP	SDF*	SHID	SIEC	TIME*	PLS	SECHDR	UIC*	COMMENT

* Indexed Field

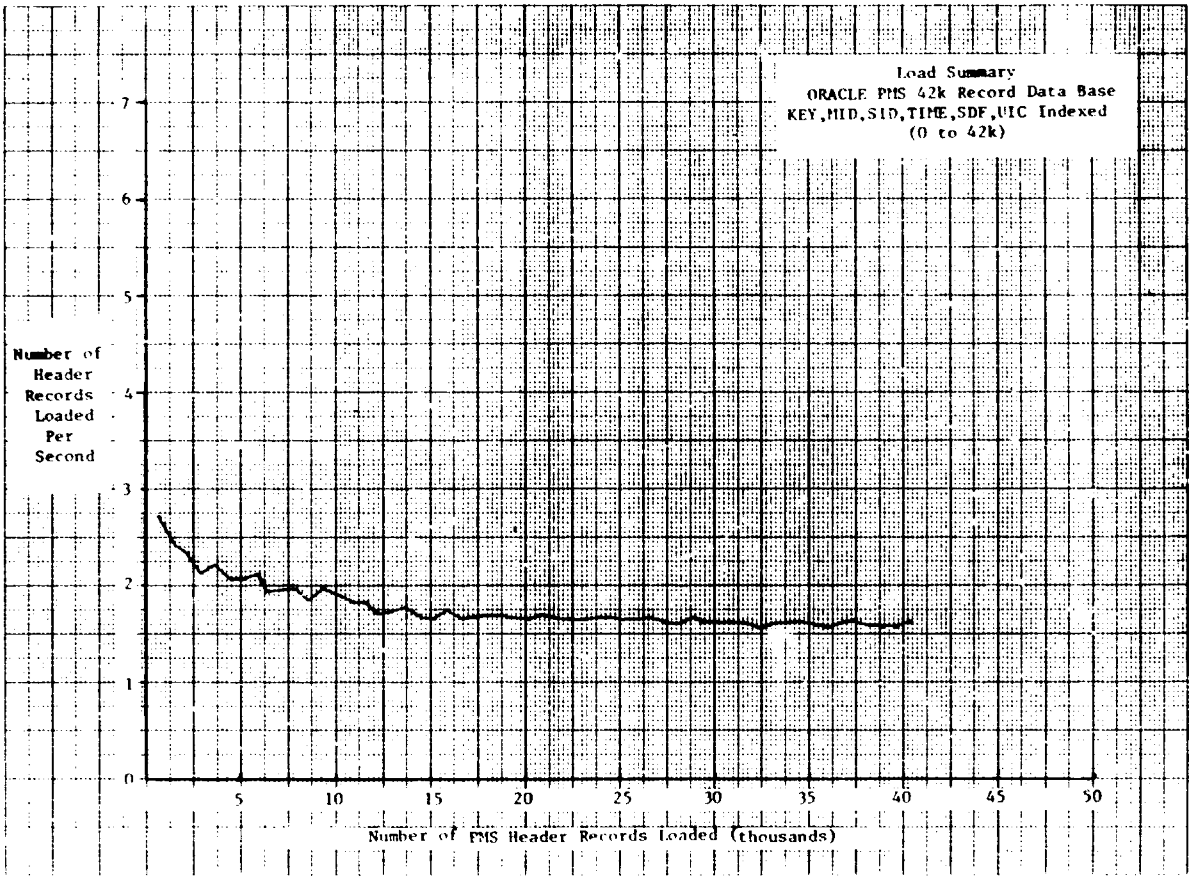
Two test loads were performed which loaded empty versions of the above table with 10,000 and 42,000 rows, respectively. The UIC and COMMENT fields are included for user created packets only; therefore in each of these tests the UIC and COMMENT fields were null in all rows. All loads were conducted without contention from other VAX/VMS users.

The 10,000 row load is summarized in the graph presented below. The format of the graph and other graphs which follow throughout Section 4 plots the average number of records loaded per second over an interval of records against the size of the data base at that time. The interval chosen is 720 records which, at the time of the testing, represented ten PMS bursts. The graphs depict the average ingestion rates observed at any particular point during the load and visually emphasize the amount of degradation present. In the graph below both of these are easily seen.



The first two points plotted (at 720 records and 1440 records) show an apparent increase in performance as the data base gets larger, while after the 1440 level a gradual degradation is observable. One might attribute this early performance increase to some initial overhead paid for opening the data base in preparation for the load, but examination of many other load results indicate that this is not the case. The observed increase in performance is atypical and might be attributed to the VAX system software activating some background task. Other than this anomaly, the graph is demonstrative of ORACLE's normally flat degradation after five thousand or so records are present. The observed range of load rates shows a high of almost 2.5 headers per second at about the 1,400 row data base size and falls to about 1.75 headers per second at the 10,000 row level.

The 42,000 row load is summarized belcw and demonstrates a more "choppy" degradation than the 10,000 record load above.

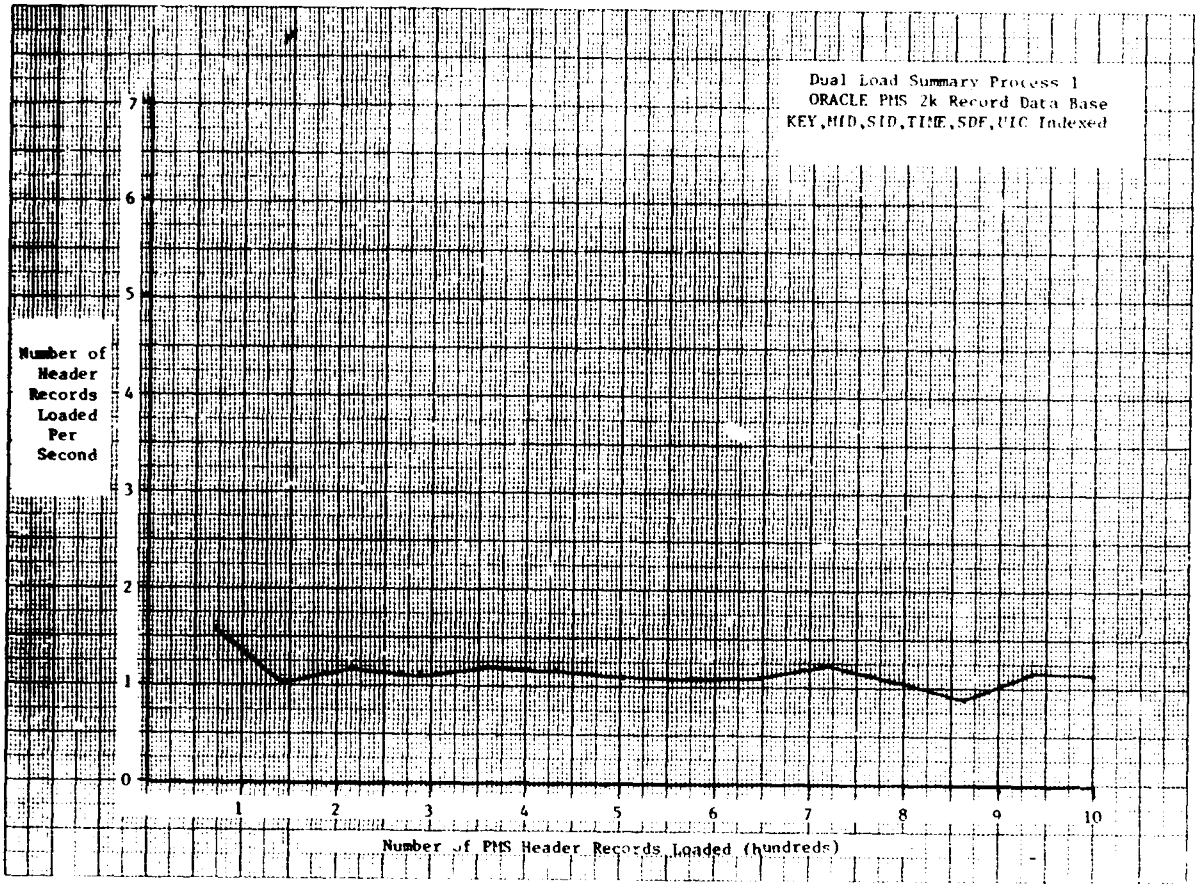


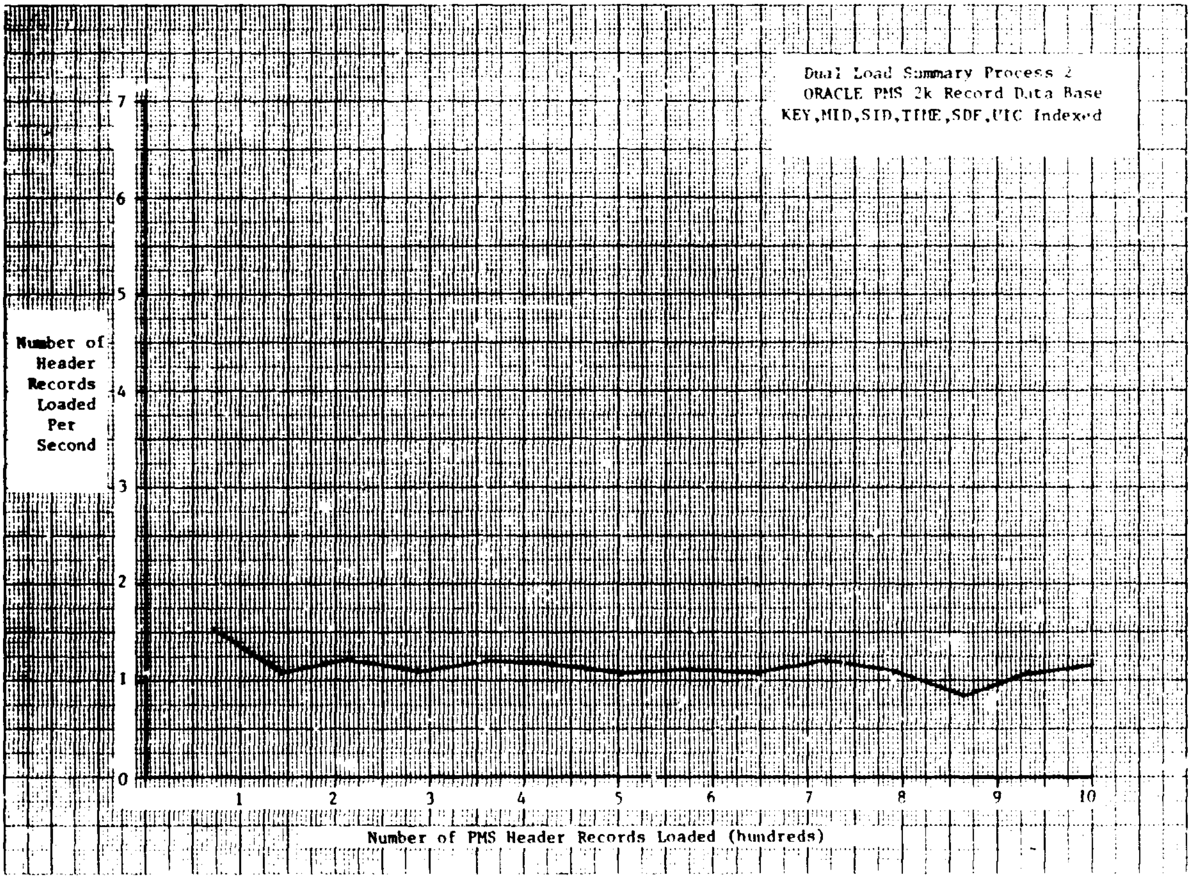
There is no initial gain in performance as seen before but rather the more typical degradation starting in a ron-linear manner and becoming very gradual shortly thereafter. The observed load rates show that initially the load rate is highest at a 2.75 header per second level but drops to the 1.85 level before reaching the 10,000 row data base size. From 10,000 rows to 42,000 rows, the load rate drops only .25 headers per second to about the 1.6 mark.

4.1.1.2 Load Performance - Dual Load

Because a PMS goal was to ingest headers at the rate of seven a second, investigation into alternative load approaches and data base designs was begun to see if an improved rate could be achieved. As one alternative approach to loading, a test was made to see if two load routines running concurrently could load more data into the same table than a single routine. The theory seemed plausible assuming that for any load process a significant amount of I/O must be present and therefore while one process is waiting for an I/O to complete, the other might be computing and initiating another I/O request. The test devised for studying this alternative required the use of two load routines (which were essentially identical), each of which loaded 1,000 PMS headers into an initially empty table. The loads were submitted simultaneously and, after completion of both, a total of 2,000 rows were present in the data base. Graphs of both loads appear on the following page and indicate that the approach is not successful in increasing total performance. When examining the graph, note that each point plotted represents a single burst, or 72 headers, rather that 720 headers as is commonly used in other graphs. Apparently, the hoped for increase in performance cannot be achieved, perhaps because ORACLE has some internal "lock out" that prevents the two load programs from achieving the synchronization theorized. An undesirable side effect of this approach is that total CPU time required rises somewhat, indicating that a greater amount of computer resources are consumed to perform the same amount of work.

ORIGINAL PAGE IS
OF POOR QUALITY



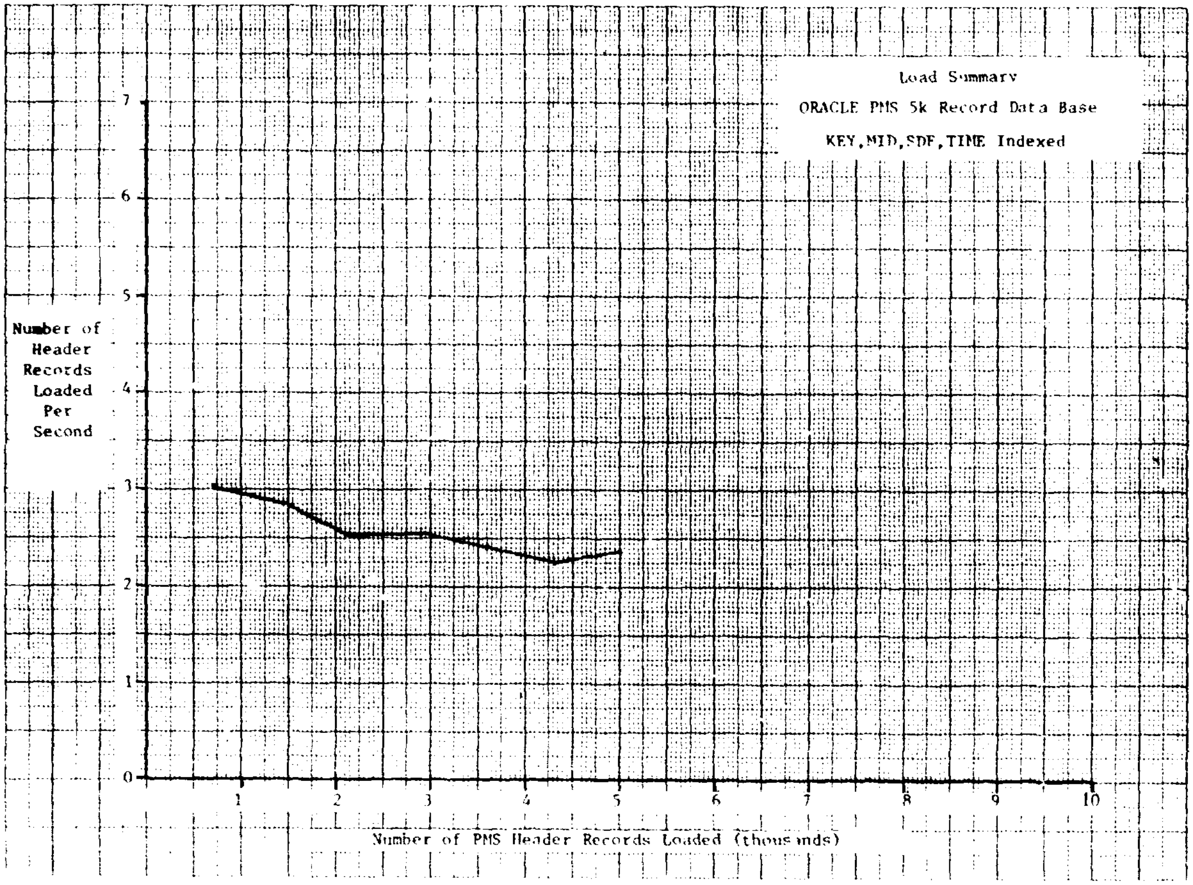


4.1.1.3 Load Performance - Reduced Number of Indexed Fields

Alternative designs of the data base were attempted which would increase load performance toward the target goal of seven headers per second. The alternatives required that features provided for in the prototype design be lost or relaxed. This is not unusual in data base design because it is seldom that a data base can serve all the needs of a diversified user group without compromise in some form. Two basic approaches were tried in altering the design. One approach tried reducing the number of indexed, or "imaged", fields to lower the overhead required to add rows to and delete rows from the data base. The trade off here is obviously the reduction of fields through which users can qualify queries and get direct access results rather than sequential access.

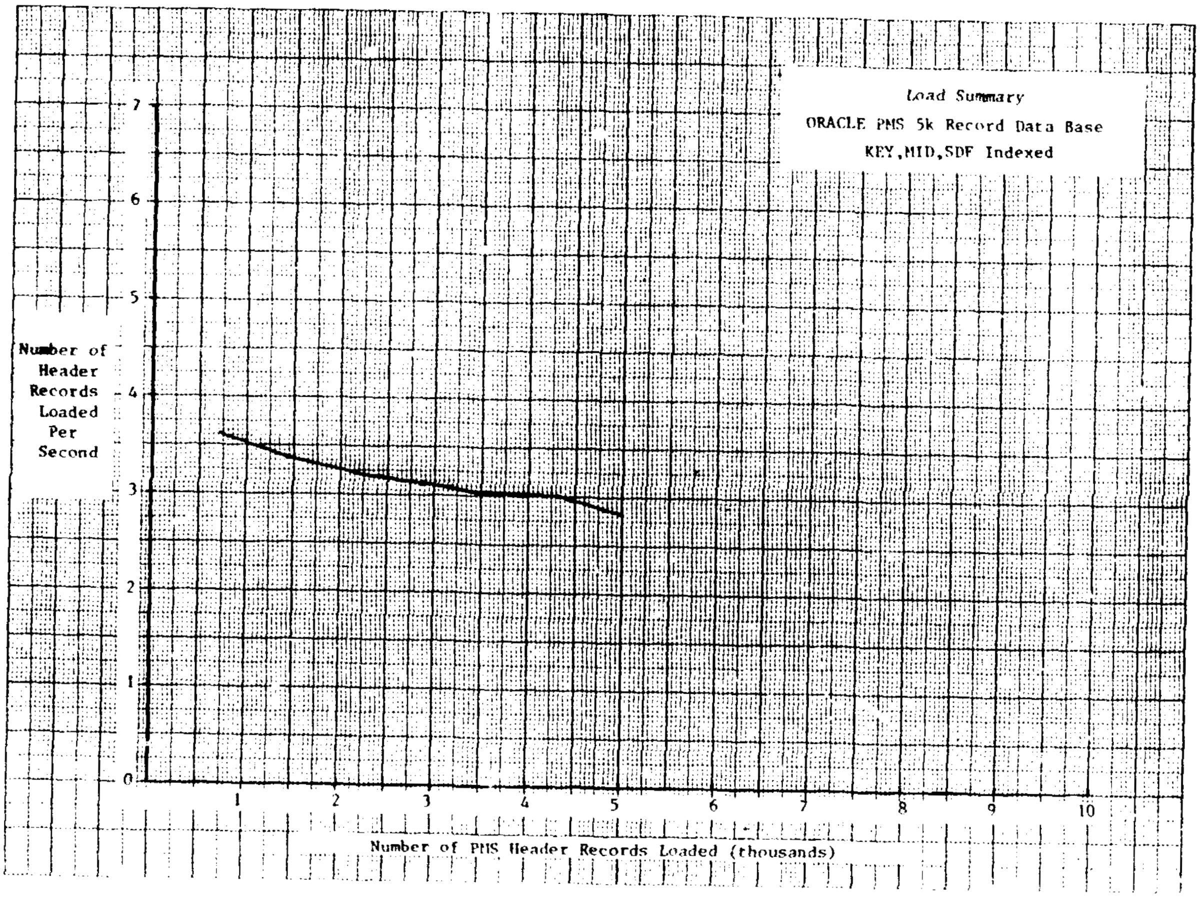
To assess the advantage (for load performance) of reducing the number of indexed fields, a series of designs were implemented and loaded with at least 5,000 records. Each design originated by eliminating a remaining indexed field from the preceding design while using the prototype design as a starting point. Since the UIC field was null, it was ignored, and, since the KEY field was required by PMS for identification of packets, it was always maintained as an index. Thus, four designs emerged as test candidates. The first eliminated the index for the SID field leaving the KEY, MID, SDF, and TIME indices. The second omitted the TIME index leaving KEY, MID, and SDF. The third omitted SDF leaving only KEY and MID and the last indexed only the KEY value.

The graph below shows the results of the first test. Here four indices, KEY, MID, SDF and TIME were present and 5,000 records were loaded with the average headers per second calculated and plotted for each group of 720 headers.



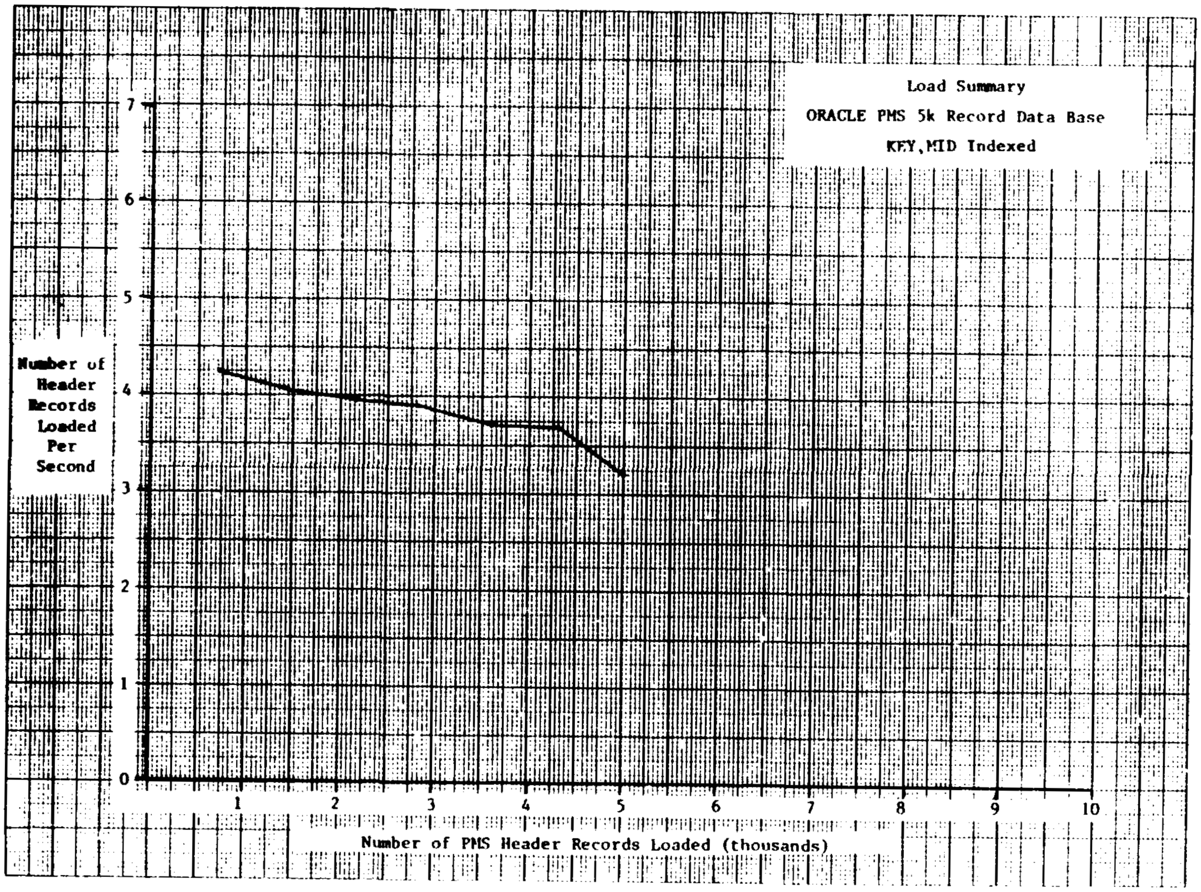
A comparison of this graph with the initial part of the graph for the 42,000 record prototype load reveals a definite improvement in performance. The initial 720 header point is the high point on both graphs and the new design shows a .25 header per second gain by beginning at a 3 header per second rate. This margin is still present at the 5,000 header point. The same gradual degradation normally observed in the ORACLE load performance can be observed as well.

By removing the TIME index from the above design another significant increase in performance can be found. A graph of the load performance using this design appears below and can be compared to the above graph. Again, 5,000 records were loaded and the results were plotted similarly.



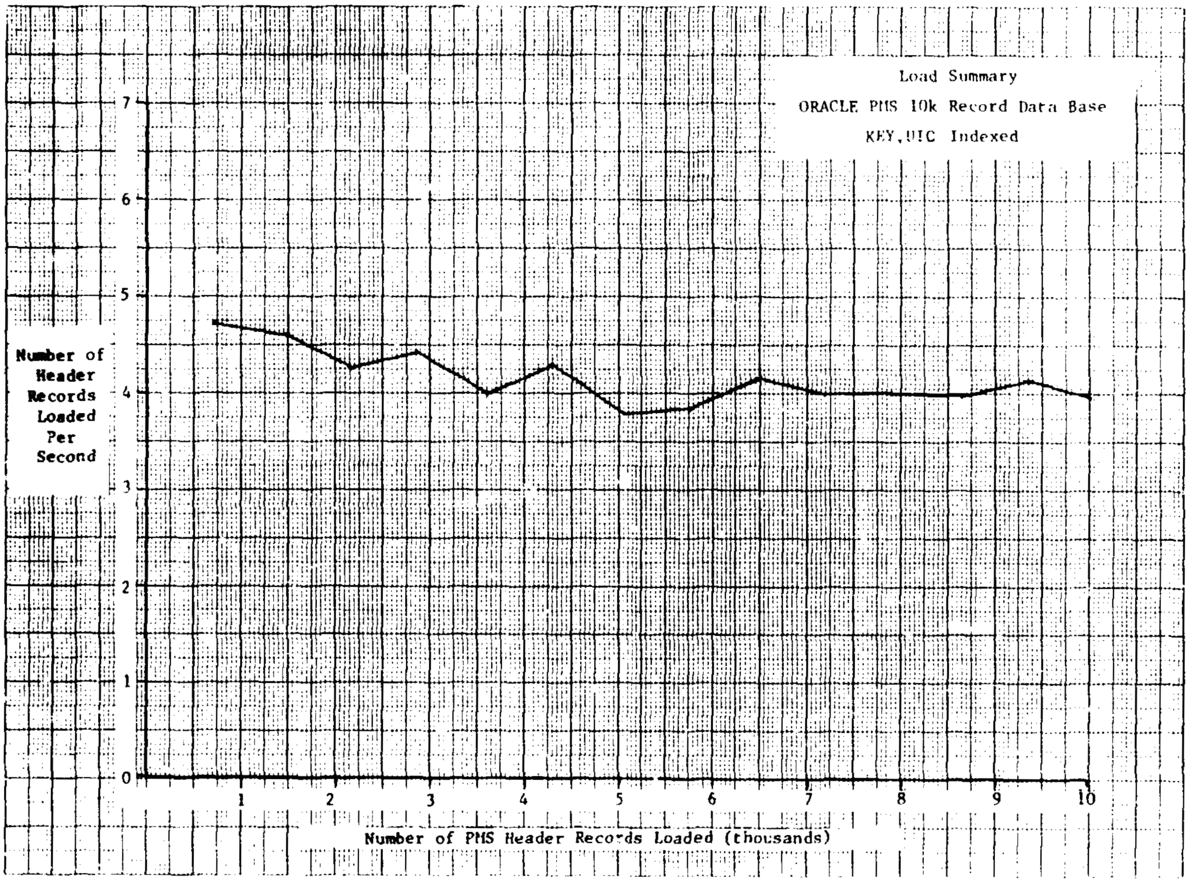
In this design, KEY, MID, and SDF were indexed. The initial load rate jumps from about 3 headers per second in the previous load to nearly 3.7 in this load. In general, a comparison of the plots demonstrates a consistent gain of from .5 to .7 headers per second throughout the load when the TIME index is absent.

The next design required indices for the KEY and MID fields, eliminating the SDF index. (Recall that when any index has been eliminated during this discussion, the field is still present as a column in the datc base table.) The results of this load have been plotted and appear in the graph below.



Another substantial gain in load performance is detectable between this graph and the preceding one when the SDF field was present. The load rate begins at about 4.25 headers per second which is more than .5 headers per second better than the previous results. An advantage of from .5 to .75 headers per second is readily observable at the same relative positions along the graph.

The last of the designs possessed the minimum single non-null index ORACLE required. Only KEY was indexed (except for the null UIC field) and, in this case, a total of 10,000 PMS headers were loaded into the data base. The results of this test are plotted on the graph below.



[illegible] 12/81

The plotted results show another gain from the previous graph. The initial point is about 4.75 headers per second, a gain of almost half a header per second. This margin is readily apparent throughout the 5,000 record level in the previous graph and continues to demonstrate the overhead that is attributable to the building and maintenance of ORACLE's B-Tree index structure.

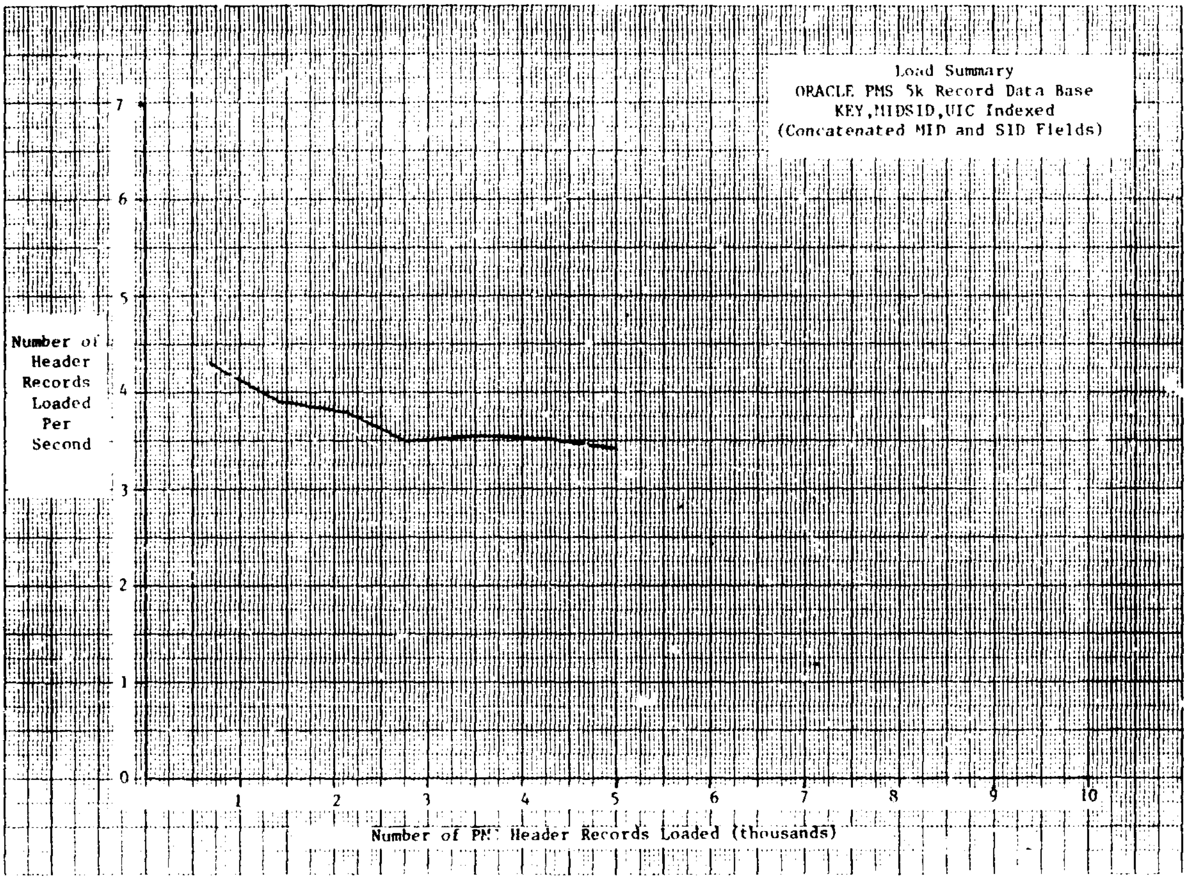
A comparison of these results, with only KEY indexed (ignoring the null field UIC), to the results of the prototype loads with KEY, MID, SID, TIME and SDF indexed, reveals an overall gain of over 2 headers per second at the 10,000 row data base size. This means that instead of loading at below 2 headers per second using the prototype design, a design with only a single index on KEY can perform at about 4 headers per second at the 10,000 record level. This is still 3 headers per second below the desired goal and would not support queries from the user community beyond users who could identify a KEY field value or values.

4.1.1.4 Load Performance - Reduced Number of Fields

A second approach in altering the data base design was to reduce the number of fields (not just the number of indexed fields), by concatenation of fields.

An examination of the logical relationships of each of the fields in the PMS primary packet header yielded a pair of fields that were good candidates for concatenation. These were the MID and SID fields. It was thought unlikely that a specification of SID independent of MID would be of any value to a user. This is because SIDs are defined uniquely for every mission; hence, no consistency need exist between SIDs of different MIDs. A test was conducted in which the MID and SID fields were concatenated into a single 16 bit field replacing the original pair. Both the KEY field and the MIDSID field were indexed (in addition to the null UIC field) and 5,000 PMS header records were loaded into an empty data base. The results of this test have been plotted on the graph below and can be compared with the graph of the test load when the KEY and MID fields were indexed that was discussed earlier in this section.

The results of both plots are somewhat similar; both start at about 4.25 headers per second while the concatenated field test concluded at about 3.4 headers per second and the other test at 3.2 headers per second. The advantage in performance which the concatenated test demonstrates is attributable to the reduction in total number of fields present in the table.



4.1.1.5 Query Performance

While, for this particular application, the query rates were not of as much interest as load rates, several queries were nonetheless performed on several of these data base designs. The queries which were performed are shown here:

```
SELECT * FROM HEADER WHERE KEY = _______
                  and
SELECT * FROM HEADER WHERE SID = _______
                       AND  MID = _______
                       AND  SDF = _______
                       AND TIME = _______
```

The results of the two queries are shown in the two tables below. Each query was executed three times. In the first, KEY, MID, SDF and TIME were indexed. In the second, KEY, MID, and SDF were indexed, and in the last, only KEY and MID were indexed. The first query was performed on 5%, or 250, of the records in the data base. Each query produced one row. The second query was performed 25 times, and multiple rows were retrieved each time (averaging 4-15 rows).

ORACLE QUERY RESULTS

SELECT * FROM HEADER WHERE KEY = ________

Indexed Fields	Average Response Time (Sec)	% Degradation in Average Response Time
KEY, MID, SDF, TIME	.204	---
KEY, MID, SDF	.199	- 2.5
KEY, MID	.239	+ 17.2

ORACLE QUERY RESULTS

SELECT * FROM HEADER WHERE SID = ___, MID = ___, SDF = ___, TIME = ___

Indexed Fields	Average Response Time (Sec)	% Degradation in Average Response Time
KEY, MID, SDF, TIME	3.25	---
KEY, MID, SDF	33.61	+ 934.2
KEY, MID	14.59	+ 348.9

In the first table, the results of the first two runs show that no real difference appeared in query results between the two, as might be expected. However, the results of the third run appear to indicate that the number of indexed fields had an impact on query performance. By looking at the results of the report file generated [illegible] the query, it was determined that the presence of an outside process was most likely the cause of the increase in average response time. This outside process artificially raised the average response time.

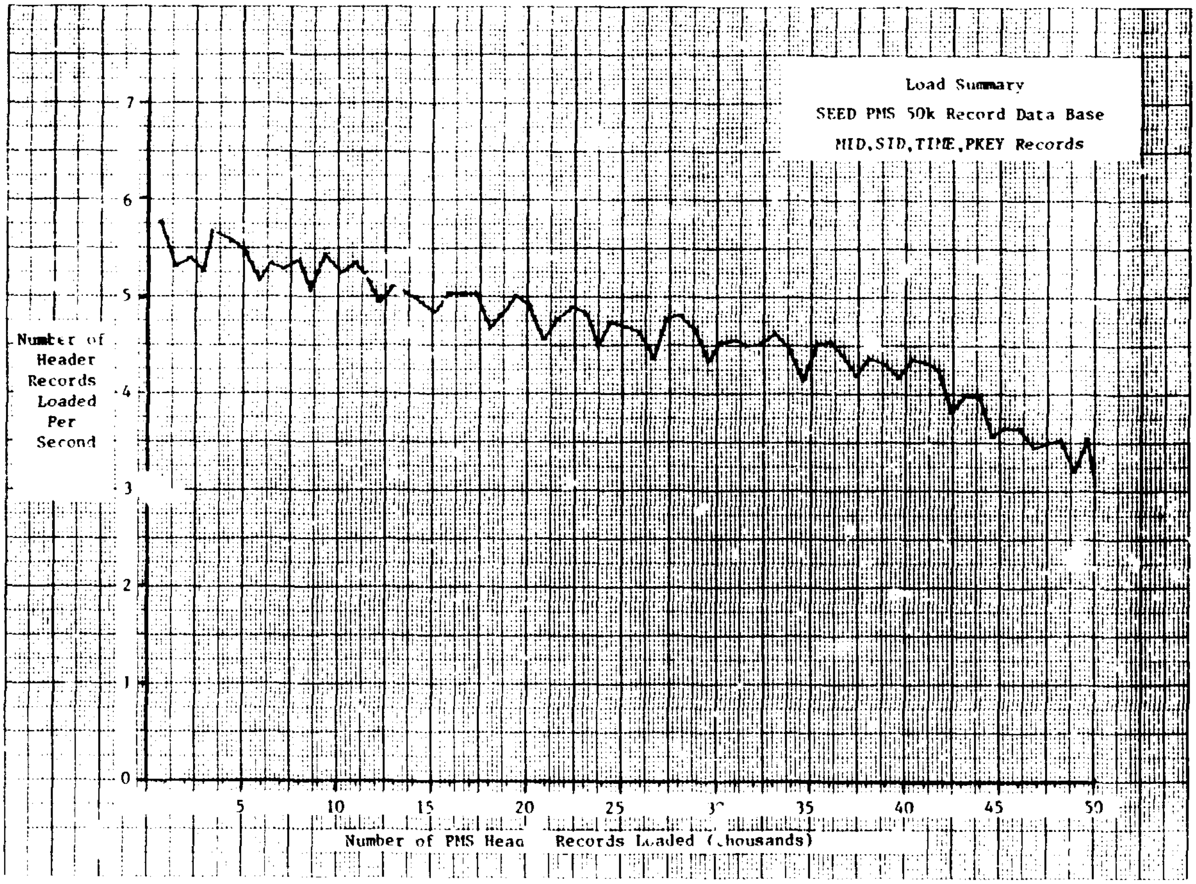
In the second table, the results of the three runs are drastically different. In the type of query executed here, the ORACLE 2.3 parsing mechanism chooses the last index encountered in the query statement as the one it uses to select rows which satisfy the query. Once all rows are selected which satisfy that part of the "where clause", a sequential search is performed to satisfy the remaining parts of the where clause. In the first run, the TIME field was the last column encountered in the where clause with an index on it. There were approximately 50 different values of TIME. In the second run, the SDF field was the last field encountered, with 4 different values. In the third run, the MID field was the last encountered, with 10 different values. When the index chosen has many unique values (as with the TIME field), for any particular value, fewer records satisfy that selection. [illegible] the TIME field, there were approximately 100 rows with each unique value of TIME. With the SDF field, there were approximately 1,250 records with each unique value, and for the MID field, there were approximately 500 records with each unique value. Therefore, in runs above, after the initial 100, 1,250, and 500 rows were retrieved for each run, respectively, a sequential search was performed on those rows to satisfy the remaining parts of the where clause. Hence the first run had the best performance and the second run had the worst performance.

4.1.2 SEED Version B.11.9

The goal of this SEED testing was to determine how a SEED data base could be altered to meet the PMS project goal of loading 7 headers per second. Also included in this section are results of queries which were performed on the various data base designs in order to gain a better understanding of SEED's query capabilities. Testing was initially performed using Version B.11, but when Version C.0 was installed, it too was tested.

4.1.2.1 Load Performance - Prototype Design

In the PMS application, the initial, or "prototype" design (see Appendix I) consisted of record R6_PKEY, which contained the entire contents of the PMS primary and secondary headers. In addition, five owner records were added. Three of these, R1_MID, R2_TIME, and R4_SID, had a significant impact on load rates. The others, R3_IID and R5_UIC were included only for user created packets, and therefore had significance on load rates only in the fact that there were pointers in the R6_PKEY record for them. A seventh record, R7_COMMENT, was also included for user created packets. R7_COMMENT is a member of R6_PKEY. Three data bases were loaded using this design, one with 50,000 header packets, one with 10,000 header packets, and one with 5,000 header packets. The load results of the 50K data base are shown on the graph below.

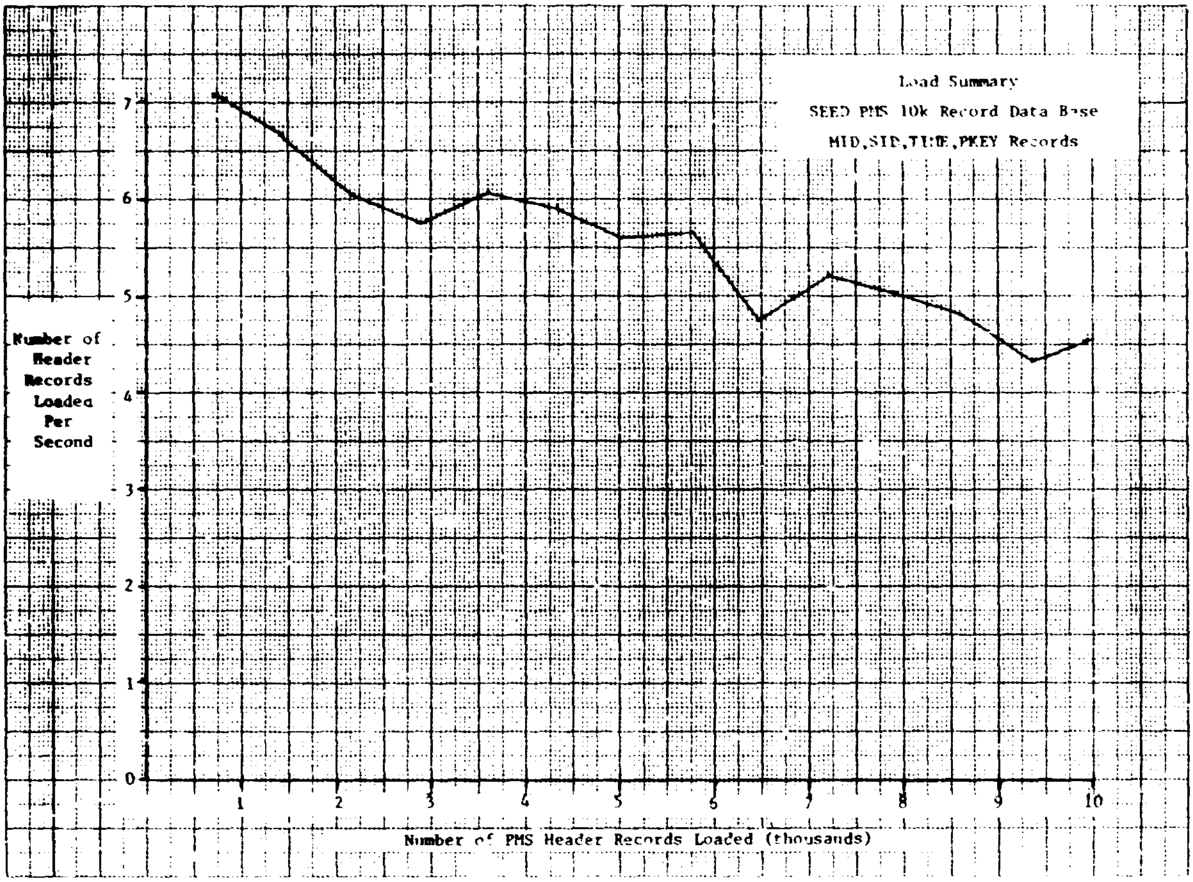


Initially the data base was being loaded at a rate of about 5.75 records per second. By the end of the load, only about 3.25 records were being loaded per second. Both of these figures fall below the original PMS project goal of 7 headers per second. At the end of the load the data base was about 85% full. A surplus area of 15% was included in the design to accommodate updates and to cut down on the number of page overflows. However, the "peaks and valleys" appearance of this graph and most of the others indicates that there were still many page overflows. A spot check of the data base statistics showed that about 40-45% of the pages in the data base were 95-100% full. Therefore, when the hashing algorithm tried to put a record on a page that was already full, a sequential search had to be done on the following pages, until space was found for the record.

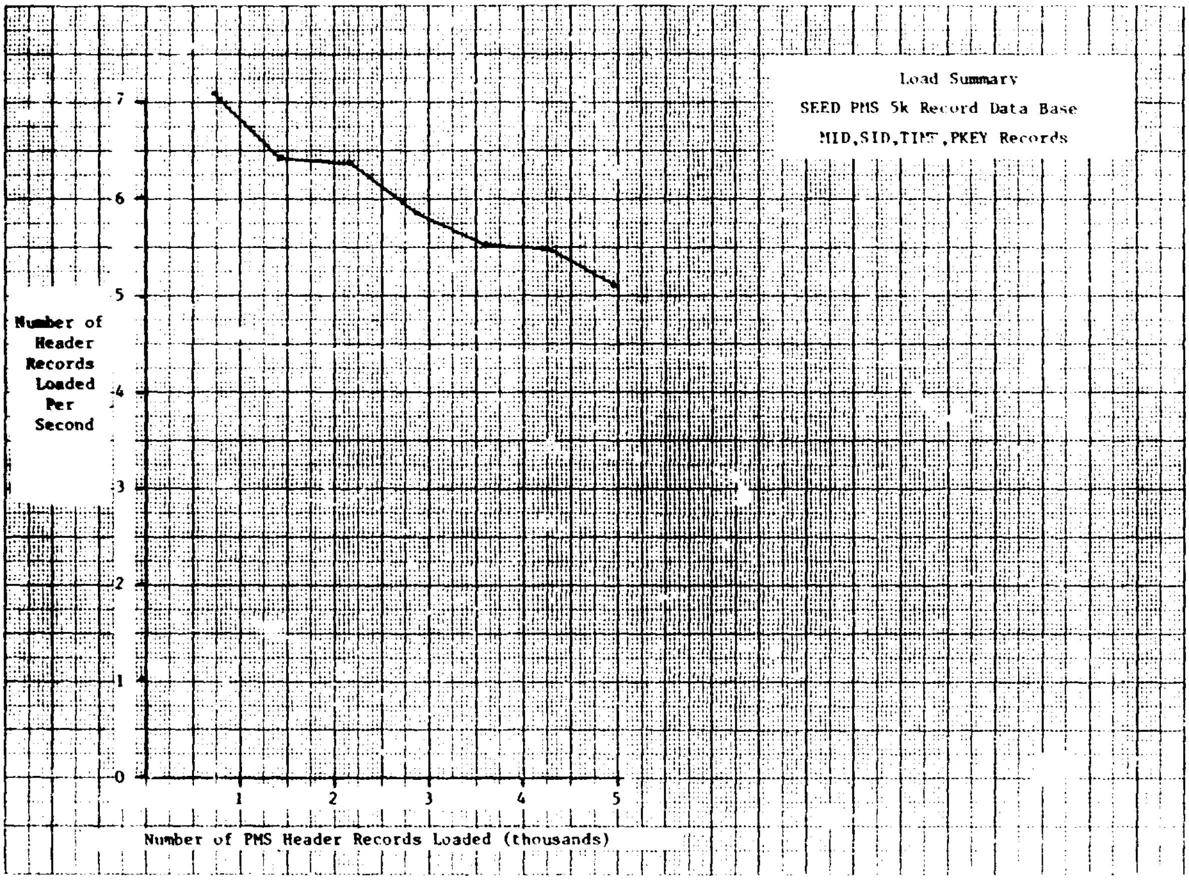
While the load rates continued to decline as the ata base became more saturated, the decline was generally linear until about 42,500 records had been loaded. At this point, the data base was about 75% full. The load rates dropped faster past this point.

In comparing these results with the 10K prototype load shown below, it is observed that the 10K load was significantly faster, from about 7 headers per second initially to about 4.5 headers per second at the end of the load.

It was thought that, at least initially, the 50K load would be faster because there were more pages available, and therefore less chance of page overflow. However, because in both runs the number of buffers available was identical. and in the 50K run there were more pages in the data base, the chance of a particular page already being in memory was reduced. This is borne out by comparing the data base load statistics in Table 4.1-1. The number of direct I/O's in the 50K load is about 5.4 times the number in the 10K load, and the total connect time is about 6 times that of the 10K load.



The 5K prototype data base shown below initially loaded at about the same rate as the 10K data base. At the end of the load, the 5K data base was about 72% full and about 5.1 records were being loaded per second. At the same point in the 10K data base load (i.e. 72% full or 8,250 records) the load rate was about 4.9 records per second, not much different than the 5K data base. However, the graph of the 5K data base load shows a somewhat smoother curve. Because of the hashing algorithm used to determine the location of records within the data base, the number of pages in the data base must be chosen carefully. The hashing algorithm uses this number in calculating a page number for a particular record. It is recommended that a prime number of pages or a number that contains large prime factors be used to aid in the even distribution of records within the data base. Two consecutive prime integers may produce very different load results. Perhaps if the next higher or lower prime integer had been chosen for the 10K load, the curve might have been smoother.


Load Summary
SEED PMS 5k Record Data Base
MID,SID,TIME,PKEY Records
7
6
5
4
3
2
1
0
Number of Header Records Loaded Per Second
1
2
3
4
5
Number of PMS Header Records Loaded (thousands)

SEED LOAD RESULTS

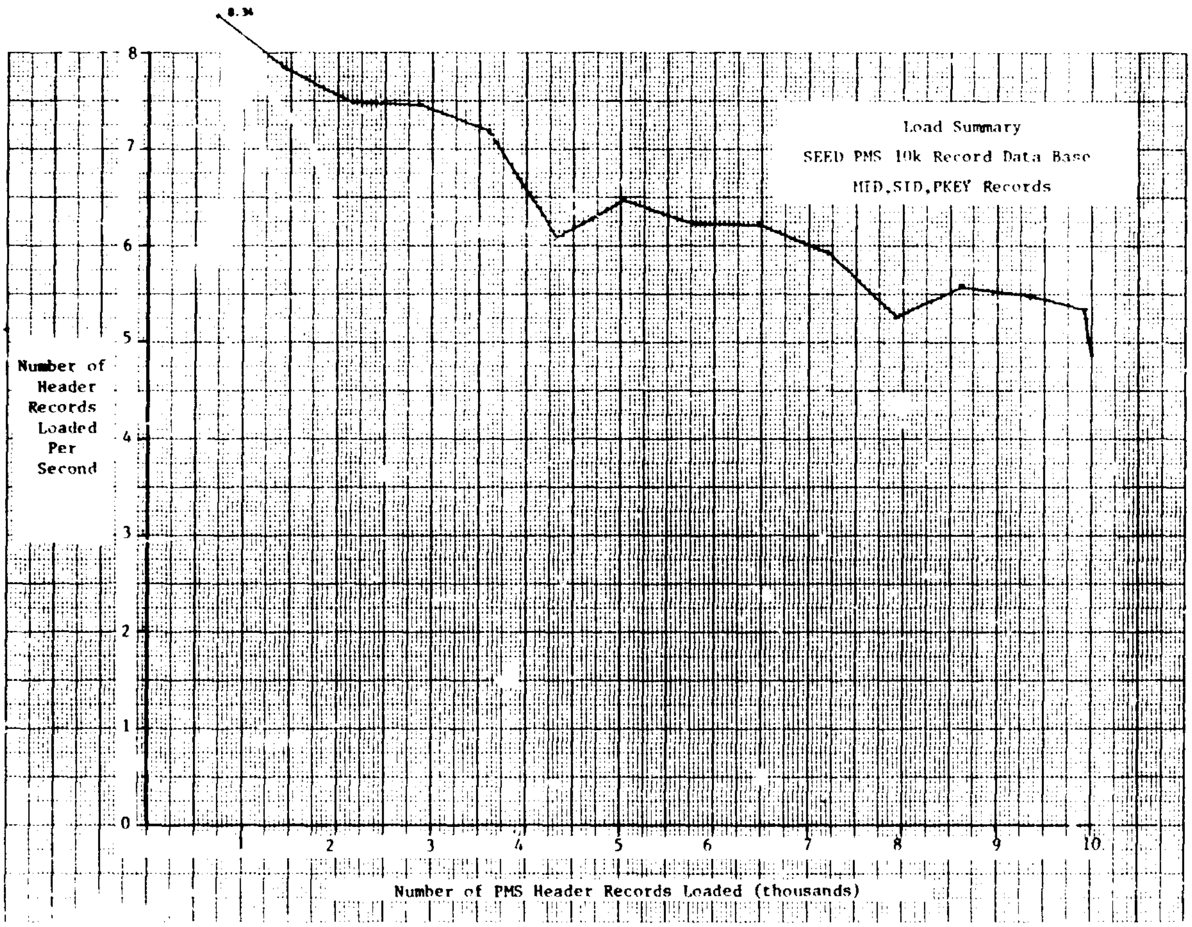
Data Base Size	Data Base Description	Average Insertion Rate (Hdrs/Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
50K	Prototype	4.55	11,000.60	4,961.82	156,354.	150,653.
10K	Prototype	5.43	1,839.96	964.68	28,771.	30,287.
5K	Prototype	5.93	843.84	450.74	12,661.	15,873.
10K	No TIME Record	6.88	1,452.47	733.70	20,927.	26,427.
5K	No TIME Record	7.61	657.35	359.49	9,732.	13,276.
10K	PKEY Record Only	8.18	1,222.48	442.32	15,352.	17,940.
5K	PKEY Record Only	12.32	405.80	208.86	7,079.	9,687.
5K	SDF Record Added	5.53	904.37	514.37	14,035.	16,033.
5K	MIDSID Combined	5.80	862.69	403.61	17,079.	17,163.

TABLE 4.1-1

4.1.2.2 Load Performance - Reduced Number of "Owner" Records

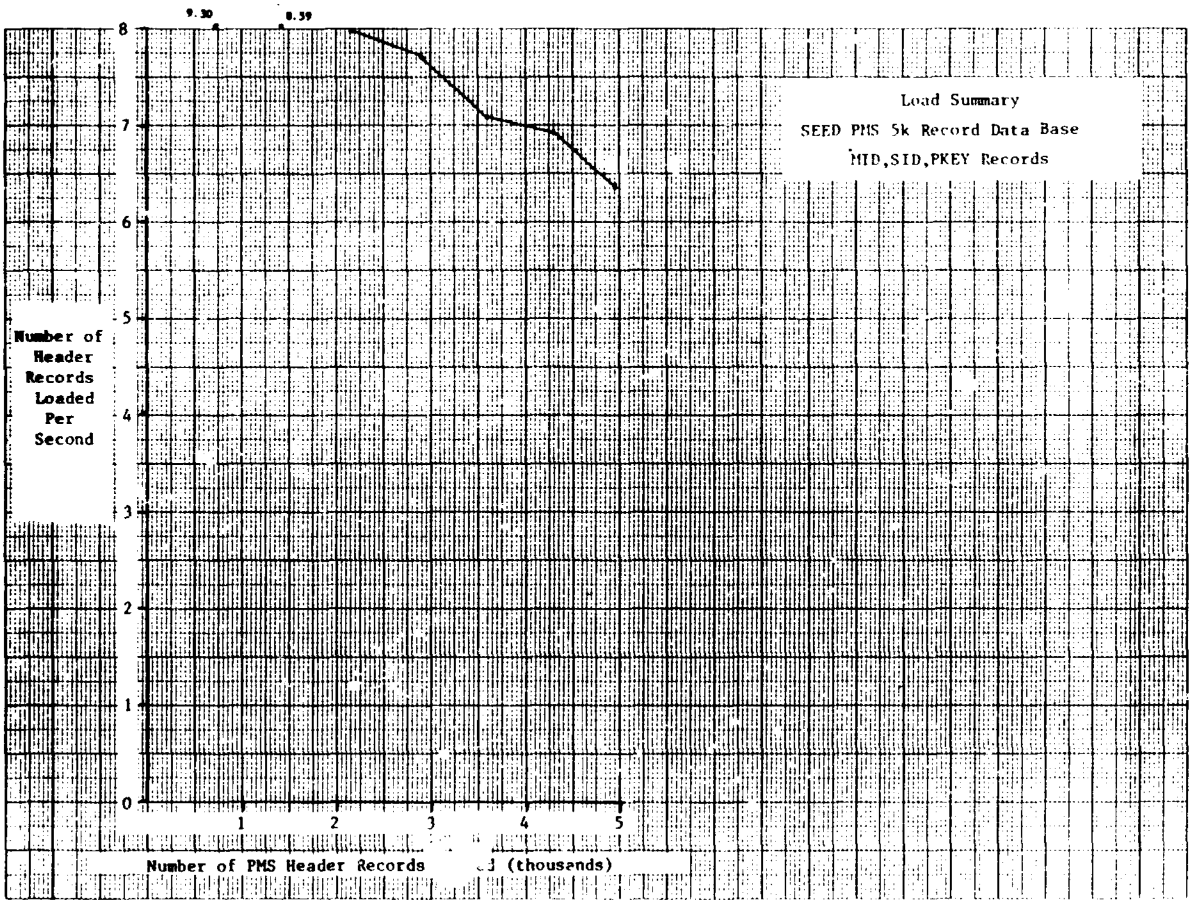
As was previously mentioned, one of the goals of the PMS project team was to load 7 headers per second. In an effort to meet this goal, data base designs were attempted which included fewer "Owner" records. The "Owner" records were included originally so that more than just the "PKEY" field could be "CALC"ed on.

The first variation was the removal of the R2_TIME record on the 10K and 5K data bases. Although initially the load rate of 8.34 headers per second on the 10K data base was well above the goal, the rate dropped off to about 5.3 headers per second, as shown in the graph below.

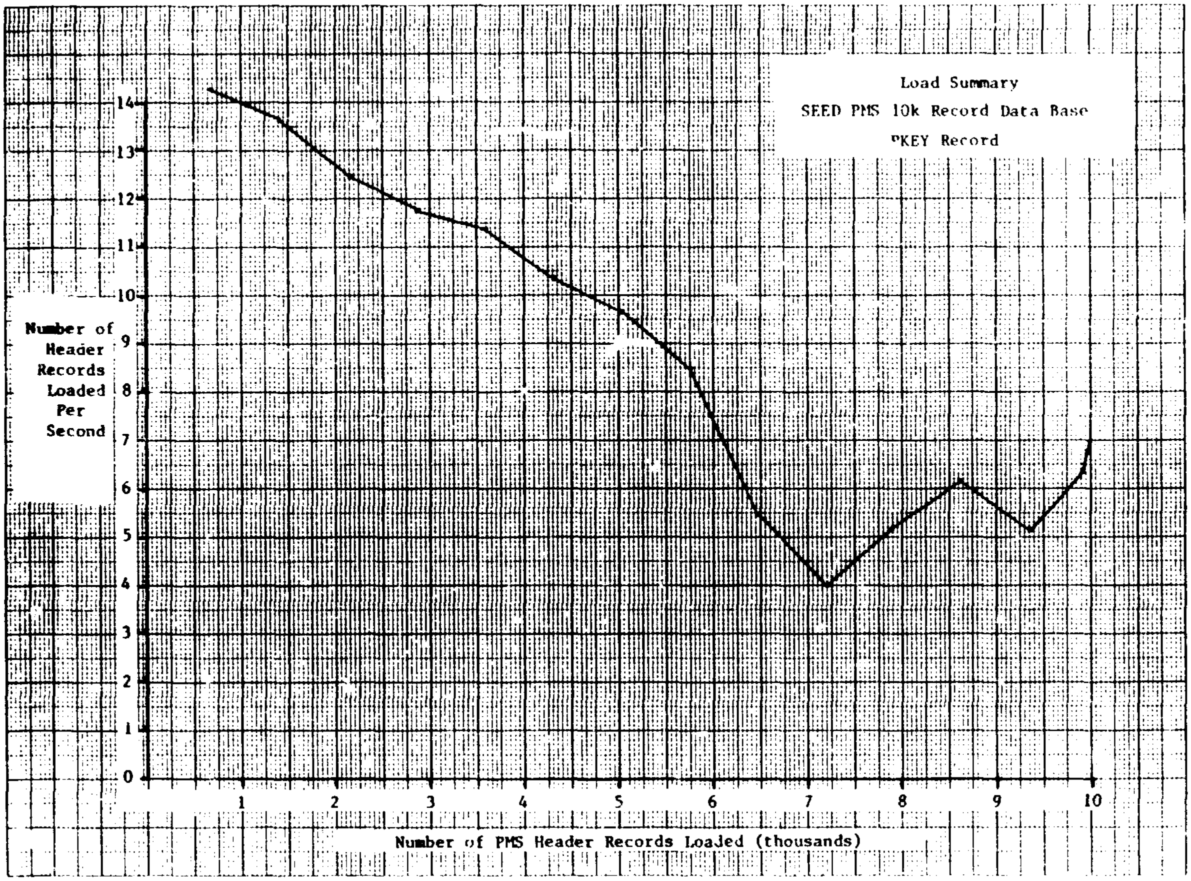


The graph indicates that there were 2 points in the load, at approximately 4,000 records and 8,000 records, where the hashing algorithm had trouble finding free space because of page overflows.

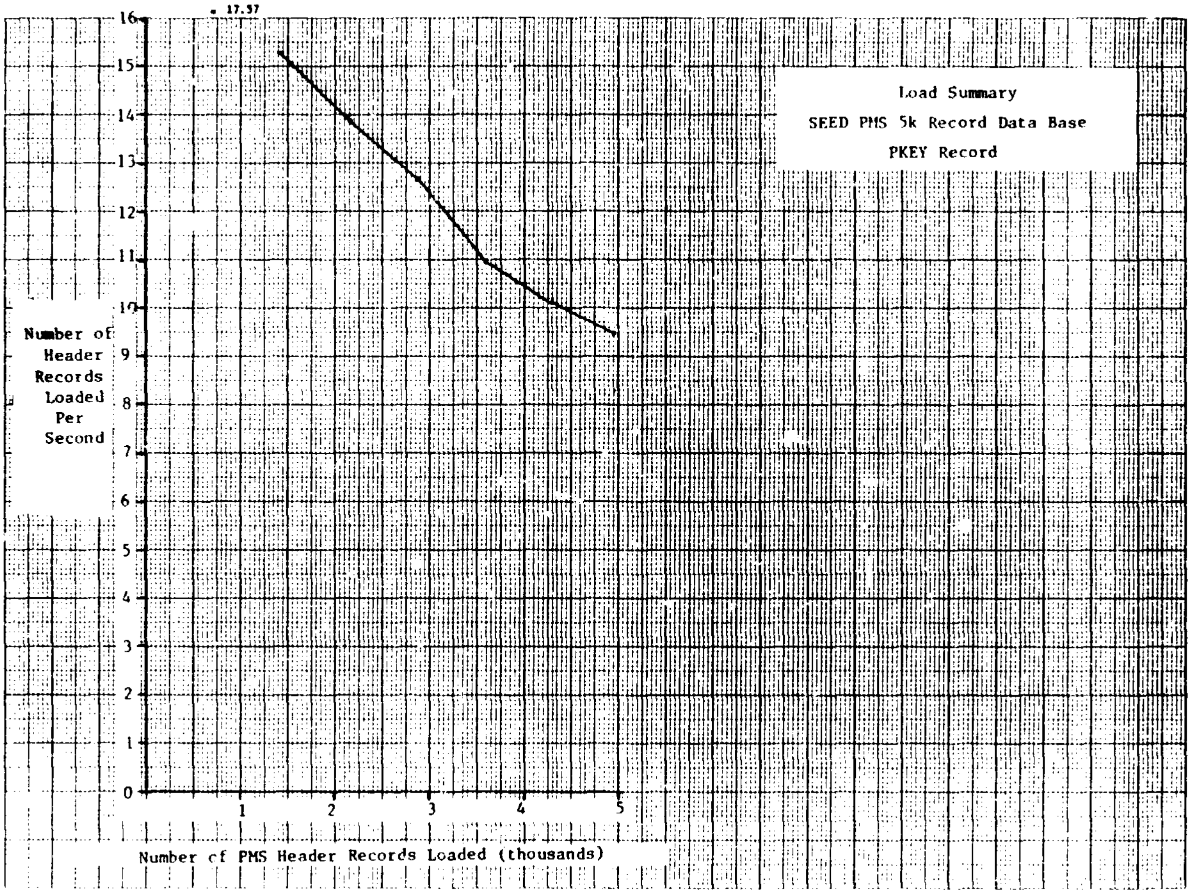
In the 5K data base load, the results improved, from 9.3 -> 6.3 headers per second, and while the load rate still did not meet the goal, the data base was loaded more evenly. At the end of the load, the data base was about 70% full, with the majority of pages 55-70% full. That graph is shown below.



The next variation on the prototype data base was to remove all records except for the R6 PKEY record. The results of the 10K data base load are shown below.

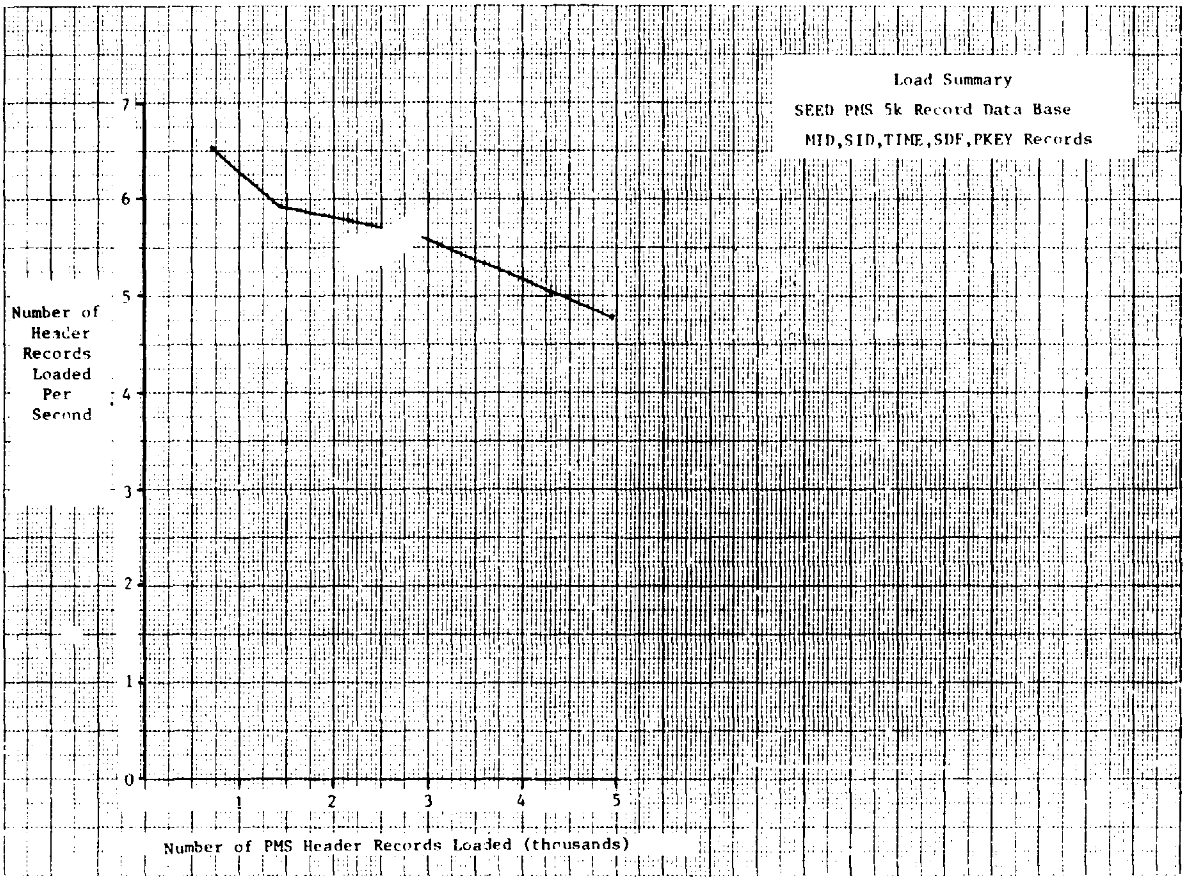


The load started at about 14.25 headers per second and declined almost linearly until about 5,750 records had been loaded. At this point there was a drastic drop from about 8.5 to about 5.5 and then on to a low of about 4 headers per second. Normally, an explanation for this would be that there were many page overflows due to poor hashing of the data. However, a glance at the data base statistics revealed that this was not the case and indeed, only 1% of the pages in the data base were 95-100%. Looking at the load statistics, there were a few times in the load where sudden jumps in time appeared. Currently, this can only be explained as an anomaly caused by VAX system software or an undetected process that interfered with the standalone mode of data base loading. In a similar case loading a 5K data base, this drastic drop in load rate did not appear (shown below). The load started at about 15.25 headers per second and ended at about 9.5 headers per second, well above the goal of 7 headers per second. The smooth curve is an indication of a well distributed data base. At the end of the load, the data base was 57% full and the majority of pages in the data base were 40-60% full.



4.1.2.3 Load Performance - Increased Number of "Owner" Records

Another data base design was implemented to test the effect of adding more "Owner" record types to the prototype design. The results in the graph below show the effect of adding one record type, R8_SDF1, to the data base design.

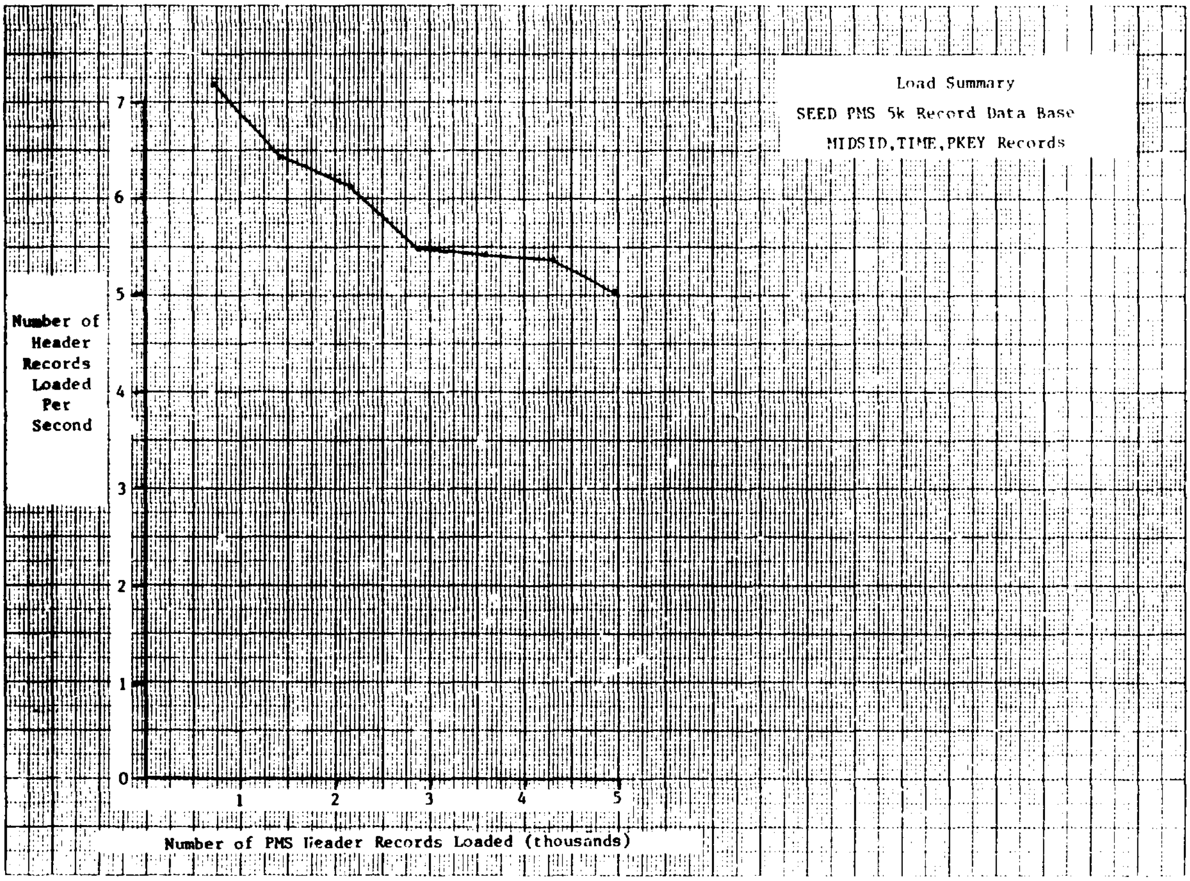


Where the prototype 5K data base loaded from 7.1 -> 5.1 headers per second, the addition of a new record type reduced these numbers to 6.5 and 4 75, respectively. Not only must the new record type be stored, but the pıoper set linkages must be made.

4.1.2.4 Load Performance - Reduced Number of Record Types

In the prototype design (Appendix I) the R1_MID and R4_SID records were stored separately; the R4_SID being stored "via" the set connecting R1_MID with R4_SID.

In the final data base design variation, these two records were combined into one. The effect of this was to reduce the number of records to be stored and to reduce the number of set linkages to be made. The results on the graph below show a slight improvement in the initial load rate from 7.1 to 7.2 headers per second, but in general the rates were about the same.



4.1.2.5 Query Performance

While a primary goal of the PMS application is the load rate of 7 headers per second, query rates are also of importance. Many tests were made to determine the effect of data base design on querying. A detailed explanaticn of each of the queries performed can be found in Appendix III. The results of performing the queries on different data base designs are discussed here. Query times include any HLI code necessary to perform the query, for example forming the unique key. Some of these results are relevant to the discussions in Section 2.3. These initial queries were performed using SEED Version B.11.9.

The first query involved accessing 5% of the R6_PKEY records in the data base by "CALC"ing directly on the unique key, PKEY, associated with the record. This unique key was formed, while loading, using a combination of 6 of the fields in the input record. In querying, a record was read from the same input file used for loading, the same unique key was formed, and the correct R6_PKEY record was accessed. This process was then repeated until 5% of the R6_PKEY records had been accessed. This is the quickest, most direct method of accessing an R6_PKEY record in the data base. However, unless the user has all 6 pieces of information available, this method cannot be used. Because this query did not depend on access to any other records in the data base, the time involved in accessing one R6_PKEY record should be independent of data base design, except for the overhead involved with pointers. The results in Table 4.1-2 show that for 5K data bases the average connect time to retrieve one record ranged from .074 to .107 seconds, or a total time difference of just over 8 seconds in accessing 250 records.

If the user does not have the necessary information to form a unique key, another method of accessing the header information must be used. The user may get to the header information by first accessing one of the other records in the data base and then doing a search on its members until the proper header is found. The results in Table 4.1-3 show that in all of the data base designs, the fastest way to retrieve a header (other than "CALC"-ing directly on the primary key) is to get to it by the owner record

SEED QUERY RESULTS - "Calc on PKEYs (5%)"

Data Base Size	Data Base Description	Average Response Time (Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
10K	Prototype	.084	21.1	12.13	367.	1,290.
5K		.074	18.4	10.65	335.	1,754.
10K	No Time Record	.107	26.8	11.77	345.	1,182.
5K		.084	21.1	10.69	326.	1,852.
5K	SDF Record Added	.081	20.2	10.62	335.	1,817.
5K	PKEY Record Only	.080	20.1	10.29	325.	1,739.
5K	MIDSID Combined	.091	22.9	10.23	326.	1,732.

TABLE 4.1-2

SEED QUERY RESULTS - "Calc_______, find PKEYs (.5%)"

Data Base Size	Data Base Description	Query Description	Average Response Time (Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
5K	Prototype	Calc MID, find SID, find PKEYs (.5%)	8.81	220.32	11.83	3,129.	6,711.
5K	SDF Record Added		6.03	150.8	58.15	3,127.	6,578.
5K	No Time Record		6.57	164.15	56.10	3,131.	6,560.
5K	Prototype	Calc MID, find PKEYs (.5%)	16.71	417.67	151.73	8,306.	10,714.
5K	SDF Record Added		14.90	372.43	153.8	8,302.	10,254.
5K	No Time Record		17.03	425.65	146.63	8,297.	10,372.
5K	Prototype	Calc TIME, find PKEYs (.5%)	3.31	82.76	30.47	1,599.	4,113.
5K	SDF Record Added		3.01	75.20	31.33	1,604.	4,346.
5K	SDF Record Added	Calc SDF, find PKEYs (.5%)	39.48	986.93	391.60	24,827.	19,251.
5K	MIDSID Combined	Calc MIDSID, find PKEYs (.5%)	7.40	185.10	55.68	3,134.	6,693.

TABLE 4.1-3

that has the most occurrences. That is, if the R6_PKEY record has two "owner" records, the R1_MID record (with 10 occurrences) and the R2_TIME record (with 50 occurrences), the fastest way to retrieve an R6_PKEY record is to find the correct R2_TIME record, and then search its members for the R6_PKEY. This is because, on the average, with 5,000 R6_PKEY records stored, each R1_MID record would have 500 members to search, but each R2_TIME would have only 100 members to search.

An interesting comparison can be made between the query "calc MID, find SID, find PKEYs" and "calc MID, find PKEYs" in Table 4.1-3. There were 10 occurrences of record R1_MID and 3 occurrences of record R4_SID for each R1_MID (or 30 R4_SID occurrences totally). In each of the three data base designs testing these two queries, the acquire time for a single header was 2 to 3 times longer using the "find MID, find PKEYs" approach than the "find MID, find SID, find PKEYs" approach. This can be explained as follows. For each R1_MID, there is an average of 500 R6_PKEY members. But for each unique combination of R1_MID and R2_SID there is an average of only 167 R6_PKEY members.

Keeping in mind that by forming a unique combination of MID and SID before searching for the proper header made access quicker, tests were made where the MID-SID combinations were formed during loading. While this did not have much effect on the load rates, query rates improved slightly.

Last, supposing that the 5,000 record data base had only one record, the R6_PKEY record, and that the user did not have enough information to form the unique key, tests were made to access the data base sequentially (FINDAP) and through the "find using" (FINDU) command. The sequential search takes the longest time, averaging 38 seconds to retrieve each header. In the "find using" access method, the user supplies the DBMS user work area with the values to be matched. The DBMS then finds the correct header. While this access method was much faster than a sequential search (averaging 19 seconds per header), it should be avoided by adding an owner record to the data base design, if possible. The following table shows the results of this testing.

C-3

SEED QUERY RESULTS

Query Description	Average Response Time (Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
Find PKEY's thru findu (.5%)	19.9	499.	360.23	4,335.	13,511.
Find PKEY's sequentially (.5%)	38.0	951.	764.93	4,334.	13,837.

4.1.3 SEED Version C.00.02

4.1.3.1 Load Performance - Prototype Design

The most significant change to the SEED DBMS in the C.0 version was the introduction of pointer arrays and indices. In the PMS-like application described in Appendix I, the C.0 testing focused on the effect of indices on loading and querying. As a guideline to comparing the B.11 and C.0 versions, the initial data base load consisted of loading the "prototype" design (See Appendix I) to determine whether the code and error handling optimization present in the C.0 version had a significant effect on load rates. In comparing the results of the B.11 and C.0 loads in the table below it can be seen that the CPU and total connect time increased slightly in the C.0 version, as did the number of page faults.

SEED LOAD RESULTS

Version	Job Number	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
B.11	1	790.	439.	12,643.	15,335.
	2	812.	437.	12,640.	15,473.
	3	822.	440.	12,646.	15,381.
C.0	1	849.	454.	12,649.	17,592.

4.1.3.2 Load Performance - Use of Pointer Arrays (Indices)

The remainder of testing with the C.0 version consisted of a data base with only the R6_PKEY record (because this was shown to load the fastest in the Version B.11 testing of variation in data base schema definition) and various indices on items in the R6_PKEY record. The loading results of these tests are shown in Table 4.1-4. Where loading a 5K "PKEY Record Only" data base (in version B.11) used 405.80 connect seconds, 208.86 CPU seconds, 7,079 direct I/O's and 9,687 page faults, the load times for all of the "indexed" runs were much greater. Because the load rate of the PMS data base is of greatest importance, the use of indices in that application may be unreasonable. However, because of the potential improvement of data base querying with the use of indices, the loading results will be discussed here. In all cases, the testing showed that loading an index where the indexed item had a low rate of duplication (or no duplication) was more desirable to one where the duplication rate was high. As an example, the 2K SID indexed run may be compared to the 5K PKEY indexed run. In the former, there are only 3 different values for SID. In the latter, since the PKEY field is unique, there are 5,000 different values. The "SID indexed" run took 1.7 times as much CPU time as the "PKEY indexed" run and only loaded 40% as much data. Of importance also in the use of indices, is the actual index specification to SEED in the schema. A comparison of the two "Time Indexed" runs indicates this. In the first, where there are 1,100 branches per node, 6 pages of 10,752 words per page, and assuming each node is half full (which is a SEED worst case), 10 nodes are needed to store 5,000 records. This can be accomplished with a two level B-tree. In the second, where there are 24 branches per node, 127 pages of 512 words per page, and assuming each node is half full, 417 nodes are needed to store 5,000 records. This can only be accomplished by going into the third level of the B-tree. The differences in the load statistics reflect these differences in index definition.

SEED LOAD RESULTS

Data Base Size	Data Base Description	Average Insertion Rate (Hdrs/Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
2K	Record R6_PKEY only; SID Indexed (1 page, pg = 10,752 wds, 500 BPN*, setbuf (20,20,18)	2.91	687.	539.	3,149.	73.
5K	Record R6_PKEY only; PKEY Indexed (5 pages, pg = 10,752 wds, 1,000 BPN*, setbuf (20,20,18)	8.83	566.	310.	9,036.	3,917.
5K	Record R6_PKEY only; Time Indexed (6 pages, pg = 10,752 wds, 1,100 BPN*, setbuf (20,20,18)	6.92	723.	440.	8,943.	1,101.
5K	Record R6_PKEY only; Time Indexed (127 pages, pg = 512 wds, 24 BPN*, setbuf (100,100,90)	5.05	990.	703.	5,922.	37,551.
5K	Record R6_PKEY only; PLI Indexed(4 pages, pg = 13,322 wds, 1,100 BPN*, setbuf (20,20,18)	7.96	628.	290.	8,942.	222.
5K	Record R6_PKEY only; MIDSID Indexed (4 pages, pg = 10,752 wds, 1,100 BPN*, setbuf (20,20,18)	5.96	839.	565.	8,931.	285.

* BPN - branches per node

TABLE 4.1-4

4.1.3.3 Query Performance - Use of Pointer Arrays (Indices)

Queries were made to try to compare the acquisition rates of R6_PKEY records between "owner-member" type access and "index" access. For example, when accessing .5% of the R6_PKEY records by use of the owner R2_TIME, member R6_PKEY method described in Appendix I, the mean CPU time was about 1.2 seconds and the mean connect time was about 3 to 3.3 seconds. Accessing the same .5% of the R6_PKEY records using an index on TIME took an average of 2.2 CPU seconds and 4 connect seconds. In indexing, a B-Tree has to be navigated to find the correct record. This takes more time than solving a formula for the data base key of the correct record. If, however, records were to be found that fell within a range of times, the B-Tree method would prove to be the faster of the two methods. The results in this Section are consistent with the findings in Section 2.3.

Last, in comparing the access times of R6_PKEY records with indexed values having little or no duplication, it can be seen that duplication presents a problem in querying as well as in loading of indices. In Table 4.1-5, where the index was on PKEY or PLI (i.e. no duplicates) the mean CPU time was about .035 seconds and the mean connect time was about .07 seconds. In the two entries where the index was on TIME (where there may be up to 100 duplicates of each TIME), the mean CPU time was about 2.2 seconds and the mean connect time was about 4 seconds. In the case of the index on MIDSID, where there were only 30 unique values, the mean CPU time was 3.7 seconds and the mean connect time was 6.4 seconds.

SEED QUERY RESULTS

Data Base Size	Data Base Description	Query Description	Average Response Time (Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
5K	Record R6_PKEY only; PKEY Indexed (5 pages, pg = 10,752 wds, 1,000 BPN*, setbuf (20,20,18)	findi PKEY on 5% of PKEYs	.067	16.64	8.75	349.	1,880.
5K	Record R6_PKEY only; PLI Indexed (4 pages, pg = 10,752 wds, 1,100 BPN*, setbuf (20,20,18)	findi PLI on 5% of PKEYs	.070	17.61	8.39	338.	1,300.
5K	Record R6_PKEY only; Time Indexed (127 pages, pg = 512 wds, 24 BPN* setbuf (100,100, 90)	findi TIME on .5% of PKEYs	3.97	99.29	54.28	1,950.	3,237.
5K	Record R6_PKEY only; Time Indexed (6 pages, pg = 10,752 wds, 1,100 BPN*, setbuf (20,20,18)	findi TIME on 5% of PKEYs	4.08	1,021.13	582.69	18,138.	80,460.
5K	Record R6_PKEY only; MIDSID Indexed (4 pages, pg = 10,752 wds, 1,100 BPN*, setbuf (20,20,18)	findi MIDSID on 5% of PKEYs	6.39	1,597.36	921.01	27,755.	115,905.

* BPN - brancnes per node

TABLE 4.1-5

4.2 DBMS Performance on Enlarged Data Base

In an effort to determine the performance capabilities of DBMSs in a larger data base environment, a test scenario was developed for the ORACLE 3.0 DBMS and the INGRES 1.3 DBMS. All testing was performed in standalone mode.

ORACLE Version 3.0 Results

In an effort to assess the performance capabilities of ORACLE 3.0 on a large data base, a PMS-like application was implemented and a total of 101,000 records were loaded into the data base. The principle purpose of this test was to determine whether problems which were apparent in an earlier version of ORACLE and documented in a report titled "Data Base Management System Analysis and Performance Testing with Respect to NASA Requirements"* had been corrected.

An explanation of the test scenario will be given here with test results to follow. The data base was first loaded with 50,000 PMS header records of the form:

HEADER Table

PRIMARY_KEY	MID_SID	TIME	UIC	SDF	MESSAGE	HEADER

It should be noted that no indices were created prior to the load. After the data base was loaded, indices were created on the PRIMARY_KEY, MID_SID, and TIME fields. Next, four queries were performed on the data base. Each one is stated below. The number of times each query was executed is given in parentheses after the query.

* E. A. Martin, R. V. Sylto, T. L. Gough, H. A. Huston and J. J. Morone, NASA Technical Memorandum 83942, August 1981.

```
Query 1:  SELECT * FROM HEADER WHERE PRIMARY_KEY = _____            (250)

Query 2:  SELECT * FROM HEADER WHERE MID_SID = _____                 (20)

Query 3:  SELECT • FROM HEADER WHERE
            TIME BETWEEN "780101000" AND "800101000"                  (5)

Query 4:  SELECT * FROM HEADER WHERE
            TIME = "781231106" AND MID_SID > -12795                   (5)
```

After the queries were completed, a job was submitted to drop all indices. Then, another 50,000 records were added to the data base. Following this, indices were created once again for PRIMARY_KEY, MID_SID, and TIME. Next, the four queries were repeated. Each query was executed the same number of times as previously. However, because there were twice as many records in the data base, each constituted half as large a percent of the total number of records in the data base as before. The indices were then dropped and query 1 was repeated. The query was executed only once, to locate a record when no indices were present. Last 1,000 more records were added to the data base and an index was created on the PRIMARY_KEY field.

The results of the three loads are shown in the table below.

ORACLE LOAD RESULTS

Records Loaded	Average Insertion Rate (Hdrs/Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
0-50K	19.0	2,625.	2,295.	179.	25,448.
50-100K	19.3	2,586.	2,278.	195.	27,202.
100-101K	19.2	52.	46.	5.	1,169.

The 1-1/2% difference in load rates was negligible and should be dismissed as small variations in the operating system. There were no problems evident in loading a large data base and no degradation surfaced as the data base grew.

Whereas in ORACLE Version 2.3, any indices to be created were done at table creation time (even though Version 2.3.2 allowed dynamic index creation and deletion), in Version 3.0 indices were created and dropped dynamically. Because the load rates depend heavily on the number of indices, in this testing, the indices were created after the data base was loaded each time.

The following table shows the results of creating indices on the PRIMARY_KEY, MID_SID, and TIME fields after 50,000 records were loaded and again after 100,000 records were loaded. It should be remembered that prior to the load of the second 50,000 records, all indices were dropped, and the results below reflect that fact. The statistics which are presented were obtained from the VAX account file.

ORACLE CREATE INDEX RESULTS

No. of Records in Data Base	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
50,000	1,622.	1,154.	21,554.	98,919.
100,000	3,324.	2,382.	43,001.	195,294.

The number of records in the data base does not seem to impact the peformance of the "create index" function in ORACLE.

The results of all querying on the data base follows. In each case, the query, number of records in the data base, and an indication of whether indices existed on PRIMARY_KEY, MID_SID and TIME is given along with a summary of the results.

ORACLE QUERY RESULTS

Query	No. of Records in Data Base	Indices	No. of Times Query Executed	No. of Responses for Each Execution	Average Response Time (Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
SELECT * FROM HEADER WHERE PRIMARY_KEY = ______	50,000 100,000 100,000	Yes Yes No	250 250 1	1 1 1	.13 .13 665.	33. 33. 665.	17. 18. 442.	601. 639. 14,286.	3,145. 3,302. 1,583.
SELECT * FROM HEADER WHERE MID_SID = ____	50,000 100,000	Yes Yes	20 20	≈1,667. ≈3,333.	64.5 128.8	1,290. 2,576.	776. 1,544.	30,844. 61,610.	10,183. 14,217.
SELECT * FROM HEADER WHERE TIME BETWEEN 780101000 AND 800101000	50,000 100,000	Yes Yes	5 5	10,019. 20,018.	340. 675.4	1,700. 3,377.	1,063. 2,109.	37,736. 75,509.	5,914. 9,335.
SELECT * FROM HEADER WHERE TIME = 781231106 AND MID_SID>-12795	50,000 100,000	Yes Yes	5 5	68. 68.	2.4 3.2	12. 16.	10. 14.	86. 83.	3,708. 12,882.

It should be noted that the average response time listed in the table is the average time to retrieve all rows in any one execution. For example, in the last row of the table, the average response time of 3.2 seconds was the average time it took to retrieve 68 rows from the header table.

In most of the cases above, the query process did not seem to be hindered by the number of rows in the data base. However, when a sequential search is done on the table because no indices exist, as in row 3 above, the number of rows to search would be instrumental in determining response time.

Finally, the time it took to drop the PRIMARY_KEY, MID_SID, and TIME indices is given in the table below at the 50,0.) and 100,000 record level. As in the create index table, the statistics presented here were obtained from the VAX accounting log.

ORACLE DROP INDEX RESULTS

No. of Records in Data Base	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
50,000	18.	10.	87.	3,555.
100,000	18.	10.	93.	3,427.

The size of the data base does not seem to have any impact on the time necessary to drop an index.

INGRES Version 1.3 Results

In an effort to determine the performance capabilities of INGRES in a large data base environment, a test scenario similar to that performed using ORACLE Version 3.0 was developed using INGRES Version 1.3. During the course of the scenario, a total of 101,000 PMS-like records were loaded into a data base table. The table was structured as:

HEADER Table

MID_SID	SSC	SDF	TIME	UIC	MESSAGE	HEADER

The test scenario was as follows. The HEADER table was initially created with no indices, as a heap structure, and 50,000 records were loaded into it using the "COPY" command. When the load was complete, the table structure was modified to ISAM on the primary key (a multifield key on MID_SID, SSC, SDF, and TIME). Secondary ISAM indices were created on MID_SID and TIME. Next, 4 queries were executed against the data base. The four queries are stated below. After each, in parentheses, is the number of times the query was executed.

```
Query 1:  RETRIEVE (HEADER.ALL) WHERE MID_SID = _______          (250)
                                AND   SSC     = _______
                                AND   SDF     = _______
                                AND   TIME    = _______

Query 2:  RETRIEVE (HEADER.ALL) WHERF MID_SID = ______            (20)

Query 3:  RETRIEVE (HEADER.ALL) WHERE
          '780101000'<TIME<'800101000'                            (5)

Query 4:  RETRIEVE (HEADER.ALL) WHERE
          TIME='781231106' AND MID_SID>-12795                     (5)
```

When the queries were completed, the two secondary indices were deleted. The table was modified to HASH and two secondary HASH keys were created on MID_SID and TIME. The same four queries were executed once again. This time, though, query 3 was executed 2 times instead of 5. The main reason for this was a concern for the amount of time the job might take. Next, the two secondary indices were deleted, and the table was modified from HASH to HEAP to prepare for another load. The load was then performed, adding another 50,000 records to the data base. The entire scenario as described above, was then repeated for the 100,000 record data base. This time, the queries were repeated 250, 20, 5 and 5 times for queries 1, 2, 3, and 4, respectively, when all indices were declared ISAM and were repeated 250, 20, 1, and 1 times for queries 1, 2, 3, and 4 when the indices were declared as HASH. After all queries had been performed on the 100,000 record data base, the secondary indices were deleted and the table was once again modified to HEAP. Query 1 was then executed one time on the non-indexed table. Using the "REPEAT APPEND" command, 10 groups of 72 records were added to the data base and then immediately deleted. The purpose of this test was to assess the performance of the "REPEAT APPEND" command. In order to test the third method of loading, 1,000 records were loaded into a temporary table with no indices using the "COPY" command. This temporary table was then appended to the HEADER table, which had a structure of HASH and secondary indices on MID_SID (HASH) and TIME (ISAM). The secondary indices were deleted, and a test was then performed to modify the table, first from HASH to HEAP, then HEAP to ISAM, then ISAM to HASH, then HASH to HEAP and finally HEAP to HASH.

The results of the loads are shown below, as reported in the VAX account log. All results include the job and the detached process which INGRES creates. In all statistics reported, the connect time, direct I/O and page faults reported is the larger of each of those for the job and detached process while the CPU time reported is the sum of the CPU times for the job and the detached process.

INGRES LOAD RESULTS

Load Method	Records Loaded	Average Insertion Rate (Hdrs/Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
COPY into HEADER table without keys	0-50K 50-100K	27.0 26.3	1,849. 1,904.	1,350. 1,344.	14,606. 14,608.	515. 513.
REPEAT APPEND into HEADER table without keys	72 Records 10 Times	3.8	188.	76.	1,667.	924.
TEMPORARY TABLE*	100-101K	.55	1,306.	789.	32,286.	1,212.

* Load to temporary table without keys, then append temporary table to HEADER table with keys.

The variation in results among methods is very significant. While the "REPEAT APPEND" and "TEMPORARY TABLE" methods may be suitable for small loads, it is obvious that use of the "COPY" command is by far the most suitable for large amount. of data. There was a small difference between the 0-50 and 50-100 results, but the 3% difference should not be regarded as evidence that the load procedure was beginning to degrade.

Because it is much more efficient to create indices on data base tables after the data is loaded, this is the procedure that was used in the testing. Mostly as a guide to what kinds of performance might be expected in creating indices, the results for both ISAM and HASH are included here, as reported in the VAX account log. It should be remembered that prior to loading the second 50,000 records, all indices were deleted, so the results listed for 100,000 records are for all of the 100,000, not just the last 50,000 records. The results include creating indices on the concatenated primary key field (modifying the table), the MID_SID field, and the TIME field.

INGRES CREATE INDEX RESULTS

Index Type	No. of Records in Data Base	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
ISAM	50,000	3,383.	1,712.	61,145.	2,567.
	100,000	7,224.	3,669.	129,068.	2,477.
HASH	50,000	5,269.	2,738.	108,434.	3,431.
	100,000	10,812.	5,874.	224,208.	3,683.

The time INGRES takes to create indices did not seem to depend on the number of records in the data base. While, in using each method, the rate was slower in the larger data base, the change in rate was small and should not be considered very significant. On the contrary, the difference in rates between the HASH and ISAM was very significant. This should just be noted, however, because the type of query to be performed on the data base should be the main determining factor over which type of index is created. The time necessary to create the index would usually be of lesser importance. As stated previously, the rates were shown here as examples of what the user might expect.

The results of all queries are shown on the following page. The table should be self explanatory as the index type, query, and other information needed to distinguish each query appear in the table along with the results. All results were obtained from the VAX accounting file.

The results of query 1 show that, as expected, the HASH method was superior in retrieving a single record on an exact match. The result of the query executed when the structure of the data base was HEAP is included to show that on a data base with a large number of records, some structure is necessary in order that retrieval time is in an acceptable range. A retrieval time of almost 5 minutes for one record would not be acceptable to most users.

INGRES QUERY RESULTS

Query	Index Type	No. of Records in Data Base	No. of Times Query Executed	No. of Responses For Each Execution	Average Response Time (Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
RETRIEVE (HEADER.ALL) WHERE MID_SID = ____ AND SSC = ____ AND SDF = ____ AND TIME = ____	ISAM	50,000	250	1	.51	128.	81.	753.	1,040.
		100,000	250	1	.55	138.	81.	786.	934.
	HASH	50,000	250	1	.47	117.	76.	620.	945.
		100,000	250	1	.46	115.	78.	672.	1,028.
	HEAP	100,000	1	1	296.	296.	144.	11,139.	760.
RETRIEVE (HEADER.ALL) WHERE MID_SID = ___	ISAM	50,000	20	≈1,667.	20.4	408.	313.	4,322.	1,310.
		100,000	20	≈3,333.	46.8	935.	628.	8,509.	1,371.
	HASH	50,000	20	≈1,667.	30.2	603.	369.	12,416.	1,383.
		100,000	20	≈3,333.	54.5	1,090.	725.	19,040.	1,815.
RETRIEVE (HEADER.ALL) WHERE '780101000'< TIME<'800101000'	ISAM	50,000	5	10,019.	8.0	40.	27.	584.	800.
		100,000	5	20,018.	14.6	73.	50.	1,079.	802.
	HASH	50,000	2	10,019.	584.5	1,169.	522.	37,401.	797.
		100,000	1	20,018.	880.	880.	392.	29,375.	762.
RETRIEVE (HEADER.ALL) WHERE TIME='781231106' AND MID_SID>-12795	ISAM	50,000	5	68.	4.2	21.	9.	559.	813.
		100,000	5	68.	4.4	22.	9.	569.	819.
	HASH	50,000	5	68.	5.2	26.	9.	554.	728.
		100,000	1	68.	7.	7.	4.	135.	759.

In the second query, even though the match was an exact one, many records satisfied the match. In this case, the ISAM structure proved to be superior to HASH. When the table is declared as ISAM in INGRES, the records are sorted on the ISAM field. A B-tree is also created. Therefore, all records satisfying the match can be retrieved more quickly. In the HASH method, the HASH must be performed to retrieve each record. Whereas the difference in CPU time between the two methods is about 17%, the number of direct I/O operations is significantly higher under the HASH structure.

In query 3, the ISAM structure was far superior than the HASH. Any time a range is given as the criteria for retrieval, the ISAM structure should be used. When a table is stored as HASH and a range query is performed, the HASH cannot be taken advantage of. A sequential search must be performed to retrieve all rows which satisfy the range.

In the fourth query, where a range and an exact match appeared, the ISAM structure proved to be slightly better than the HASH. Perhaps the query should have been executed more times to get a better handle on how much better the ISAM structure was. Both the MID_SID and TIME fields had secondary indices of ISAM on them. While an ISAM index is best for ranges, it still performs well for an exact match. The same is not true for HASH, as was seen in query 3. HASH performs well on exact matches but not on ranges.

In all the query results discussed above, the size of the data base did not impact the performance significantly. This would not be the case, however, in a HEAP structure. In that case, the size of the data base would be the most important factor in query performance.

The results of deleting the secondary indices, MID_SID and TIME, and modifying the table to HEAP from a HASH on the primary key are given in the table below at the 50,000 record level and at the 100,000 record level. These results were obtained from the VAX accounting log.

INGRES DELETE INDEX RESULTS

Index	No. of Records in Data Base	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
MID_SID	50,000	6.	3.	41.	447.
TIME	50,000	6.	3.	45.	484.
Primary Key	50,000	812.	173.	25,075.	533.
MID_SID	100,000	5.	3.	45.	488.
TIME	100,000	5.	3.	41.	451.
Primary Key	100,000	1,638.	354.	51,993.	534.

When a command is issued to "DELETE" a secondary index, the rows are deleted from the index, which is itself a table. The index (table) is not actually dropped until a "MODIFY" is executed on the primary table.

The performance of the "MODIFY" was not degraded by a larger number of records in the data base. When there were twice as many records present, the "MODIFY" took twice as long.

The final test which was performed on the large data base was to modify the structure of the data base table to attempt to determine the kind of performance a user might expect in doing so in a real situation. In the test, there were 100,000 records in the table. Initially, the table was defined as HASH on the four fields which make up the primary key (MID_SID, SSC, SDF, and TIME). It was "MODIFY"ed first to HEAP, then from HEAP to ISAM, then ISAM to HASH, HASH to HEAP, and finally HEAP to HASH. All results are shown below as reported in the VAX account log.

INGRES MODIFY RESULTS

Modify	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Page Faults
HASH to HEAP	1,690.	364.	52,257.	540.
HEAP to ISAM	3,358.	1,517.	88,323.	837.
ISAM to HASH	4,327.	1,611.	130,184.	813.
HASH to HEAP	1,626.	350.	52,295.	505.
HEAP to HASH	4,072.	1,480.	128,792.	772.

In a MODIFY to HEAP, the structure is removed from the table, but no reordering is performed on the records, so modifying to HEAP appears as the quickest in the above table. In modifying from HEAP to ISAM, the rows must be scrted. This is accomplished by the use of temporary tables. In modifying from HEAP to HASH, the location of each record is determined by the use of an INGRES hashing algorithm, which appears to take longer than sorting for the ISAM structure. In modifying from ISAM to HASH, the same algorithm must be performed for determining the placement of each record.

While the times vary greatly between the three structures, the main concern would not usually be the performance of the "MODIFY", but instead the performance of querying or updating the table. The results were given only as an aid to the user.

4.3 Basic Load and Query

Using the PCDB application as described in Appendix II of this report, a basic data base load and query were performed. The test was conducted using ORACLE 2.3, ORACLE 3.0, SEED C.0, INGRES 1.3 and INGRES 1.4.

Input records in the PCDB application were divided into 4 types - tape level, file level, item level, and cat level. There were 55 tape level records, 13,501 file level records, 20 item level records, and 55 cat level records, for a total of 13,631 input records.

All tests conducted in this section were run in a standalone environment.

ORACLE Version 2.3.2 Results

The ORACLE data base design is discussed in Appendix II of this report. The 13,631 input records were stored in 5 tables.

The results of loading the data base are given here. All results include the detached process as well as the job, as obtained from the VAX accounting file.

ORACLE LOAD RESULTS

Average Insertion Rate (Hdrs/Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
2.74	4,976.	3,037.24	13,529.	65,621.	28,666.

The query which was executed against the data base was:

```
SELECT CAT, CATEGORY, FUNCTION, FILE.FILENUM, FLSTART, FLSTOP,
       FLFIRSTORB, FLLASTORB, FLLEN
FROM FILE, CAT
WHERE FILE.TAPEID = CAT.TAPEID AND
      [CAT.FILENUM = FILE.FILENUM OR CAT.FILENUM = NULL] AND
      FILE.NUMITEMS = 0 AND
      CAT = <'OZONE'> AND
      [FLSTART <='710401000000' AND FLSTOP> = '700801000000'] AND
      FILE.TAPEID =<'DPFL0001', 'DPFL0002', ......, 'DPFL0016'>;
```

A total of 2,796 rows were retrieved from the data base. The results appear in the following table, as obtained from the VAX accounting file.

ORACLE QUERY RESULTS

Average Response Time (Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
.177	494.	392.57	4,260.	5,737.	1,829.

ORACLE Version 3.0 Results

The PCDB application is described fully in Appendix II of this report. A brief description of the data base design will be given here. Five tables were designed to hold the PCDB data - the TAPE table, FLE table (the term FLE was used because FILE is an ORACLE reserved word), ITEM table, CAT table, and ITEM_DESCR table. Because in the PCDB application, the query response rate was of more importance than the load rate, testing was concentrated in that area. However, for the purpose of comparing ORACLE's data base load rate with that of other DBMSs, the results will be given here. There were 13,631 input data records - 55 tape level, 13,501 file level, 55 cat level, and 20 item level records. After the data base was loaded, there were 55 TAPE records, 13,501 FLE records, 20 ITEM records, 55 CAT records and 20 ITEM_DESCR records.

A summary of the loading results appears below, as reported in the VAX accounting log.

ORACLE LOAD RESULTS

Average Insertion Rate (Hdrs/Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
4.68	2,914.	2,442.	12,225.	137.	57,412.

Because, at the time of writing of this document, the ORACLE 3.0 query optimization was not complete, those results do not appear here.

SEED Version C.00.03 Results

SEED Version C.0 was also tested using the PCDB application described in Appendix II. Whereas the load rates were the major concern in the PMS-like application, query rates were given higher priority in the PCDB application. First, a data base was loaded. There were 55 input tape records, 13,501 file records, 20 item records and 55 cat records. While the load rates are not of great concern, the load statistics will be given here as reported in the VAX accounting file. Their usefulness may be more in their comparison with other data base management system results, rather than in the statistics themselves.

SEED LOAD RESULTS

Average Insertion Rate (Hdrs/Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
9.05	1,507.	1,326.	2,088.	109.	643.

Next, a query was performed on the data base. The query was complex, making it necessary to navigate a large portion of the data base. The code which performed the query is shown on the following page. A comparison of the code and the data base schema represented in Figure II.1 shows the navigation which took place in the query. In the query, all unique combinations of CAT, CATEGORY, FUNCTION, FILENUM, FLSTART, FLSTOP, FLFIRSTORB, FLLASTORB, and FLLEN were retrieved where the CAT was 'OZONE', the TAPEID was 'DPFL0001', 'DPFL0002', ... or 'DPFL0016', the file start time was less than or equal to '710401000000', the file stop time was greater than or equal to '700801000000', and there were no items present in the file.

```
      PARAMETER DSPLY=6
      INCLUDE 'DBA1:[DBMSTEST.SEEDPCDB]PCDBF.WRK'
      CHARACTER TAPES(16)*15,START*13,STOP*13
      INTEGER*2 NUMCATS
      INTEGER*4 START_CONNECT,STOP_CONNECT,CONNECT
      INTEGER*4 START_CPU,STOP_CPU,CPU
      INTEGER*4 START_DIRIO,STOP_DIRIO,DIRIO
      INTEGER*4 START_PGFLTS,STOP_PGFLTS,PGFLTS
      DATA TAPES/'DPFL0001','DPFL0002','DPFL0003','DPFL0004',
     1                'DPFL0005','DPFL0006','DPFL0007','DPFL0008',
     2        'DPFL0009','DPFL0010','DPFL0011','DPFL0012',
     3        'DPFL0013','DPFL0014','DPFL0015','DPFL0016'/
      DATA START/'710401000000'/,STOP/'700801000000'/
      CALL TIMERS(START_CONNECT,START_CPU,START_DIRIO,START_PGFLTS)
      CALL INIT(DSPLY)
      KOUNT=0
      ERRSTA=0
      CAT='OZONE'
      CALL OBTNC(R4_CAT,'FIRST')
      IF(ERRSTA.NE.0)CALL ERROR(DSPLY)
      DO 20 I=1,16
              TAPEID=TAPES(I)
              CALL OBTNC(R5_TAPE,'FIRST')
              IF(ERRSTA.NE.0)CALL ERROR(DSPLY)
              CALL OBTNPO('FIRST',S4_6)
 5            IF(ERRSTA.EQ.0307)THEN
                      ERRSTA=0
                      GO TO 10
              ENDIF
              IF(ERRSTA.NE.0)CALL ERROR(DSPLY)
              CALL OBTN(R7_LINK_TAPE_CAT,'FIRST')
              IF(ERRSTA.EQ.0326)THEN
                      ERRSTA=0
                      CALL OBTNPO('NEXT',S4_6)
                      GO TO 5
              ENDIF
              IF(ERRSTA.NE.0)CALL ERROR(DSPLY)
 6            CALL OBTNPO('NEXT',S5_10)
              IF(ERRSTA.EQ.0307)THEN
                      ERRSTA=0
                      GO TO 10
              ENDIF
              IF(ERRSTA.NE.0)CALL ERROR(DSPLY)
              IF(NUMITEMS.NE.0)GO TO 6
              IF(FLSTART.GT.START)GO TO 6
              IF(FLSTOP.LT.STOP)GO TO 6
              NUMCATS=CNTMEM(S5_7)
              IF(NUMCATS.GT.1)THEN
                      CALL OBTN(R15_LINK_FILE_CAT,'FIRST')
                      IF(ERRSTA.EQ.0326)THEN
                              ERRSTA=0
                              GO TO 6
                      ENDIF
              ENDIF
              IF(ERRSTA.NE.0)CALL ERROR(DSPLY)
              KOUNT=KOUNT+1
              GO TO 6
10    CONTINUE
20    CONTINUE
      CALL TIMERS(STOP_CONNECT,STOP_CPU,STOP_DIRIO,STOP_PGFLTS)
```

The query was submitted two times, with 2,796 responses to the query each time. The results of the two queries are given in the table below. The statistics are from the VAX accounting file.

SEED QUERY RESULTS

Query Job	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
1	.020	---	56.86	48.29	240	51	238
2	.022	+ 10.0	61.56	50.09	240	51	238

There is a 10% difference in average response time between the two runs but only a 3.7% difference in CPU time between the two. Because the number of direct I/O's and page faults are identical between the two runs, an explanation for the variation might be the presence of an outside process in the system,or just operating system variation.

INGRES Version 1.3 Results

In the PCDB application, the data base was first loaded with 13,631 records into five tables. The tables are described further in Appendix II. There were 55 tape records, 13,501 file records, 20 item records, and 55 cat records. While the load statistics are of lesser importance in this application than in the PMS-like application, they will nonetheless be given here for comparison with the other DBMSs and with a newer version of the same DBMS. All results include the detached process as well as the job, as obtained from the VAX accounting file. The two methods of loading, copy and repeat append, are discussed in Section 3.2.1.3.4 of this report. The statistics are repeated here for purposes of comparison.

INGRES LOAD RESULTS

Type of Load	Average Insertion Rate (Hdrs/Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
Repeat Append	3.3	4,105.	1,716.	30,511.	97,116.	4,340.
Copy	31.6	431.	324.	3,122.	1,451.	514.

The query which was performed against the data base was:

```
RANGE OF F IS FILE, C IS CAT
RETRIEVE (C.CAT, C.CATEGORY, C.FUNCTION, F.FILENUM, F.FLSTART,
         F.FLSTOP, F.FLFIRSTORB, F.FLLASTORB, F.FLLEN)
WHERE    F.TAPEID = C.TAPEID AND
         (C.FILENUM = F.FILENUM OR C.FILENUM = 0) AND
         F.NUMITEMS = 0 AND
         C.CAT = 'OZONE' AND
         (F.FLSTART <='710401000000' AND F.FLSTOP >='700801000000') AND
         F.TAPEID = 'DPFL*'
```

The results of the query are shown below. There were 2,796 rows which satisfied the query and were retrieved. The statistics are reported from the VAX account file and include statistics for both the host process and the detached process.

INGRES QUERY RESULTS

Average Response Time (Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
.053	148.	92.	2,189.	4,588.	1,205.

INGRES Version 1.4 Results

The data base design used in this testing was identical to that used in the INGPES 1.3 testing discussed prior to this and reported in Appendix II.

The data base was loaded using only the copy method and those results appear, as reported in the VAX account log, below.

INGRES LOAD RESULTS

Average Insertion Rate (Hdrs/Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
31.9	427.	327.	3,121.	1,451.	570.

The query which was executed was identical to that given for the INGRES 1.3 testing just prior to this. The results are given here as reported in the VAX accounting log.

INGRES QUERY RESULTS

Average Response Time (Sec)	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
.054	151.	94.	2,177.	4,538.	1,240.

4.4 Predicate Reordering

A test was conducted to determine whether the order of the various parts of the where clause in a complex query had an impact on the rate of retrieval. This test was run using the ORACLE 2.3 DBMS and the INGRES 1.4 DBMS and the PCDB application was used as the testbed. This application is described in detail in Appendix II.

ORACLE Version 2.3.2 Results

A PCDB data base was loaded with 13,631 input records into 5 tables. The ORACLE data base design is discussed in Appendix II. A test was conducted to determine the effect on query rates of varying the order of the different parts of the where clause.

The number of successful responses returned was governed by restricting the time range in the query. Two ranges were employed in the testing. The first, FLSTART<='710401000000' and FLSTOP >= '700801000000', eliminated about half of the possible responses for a total of 2,796 rows, and the second, FLSTART <= '710201000000' and FLSTOP >= '701001000000', eliminated about three fourths of the possible responses, yielding 1,412 rows.

The basic query is shown here, with the numbers to the left designating the various phrases of the where clause.

```
          SELECT CAT, CATEGORY, FUNCTION, FILE.FILENUM, FLSTART, FLSTOP,
              FLFIRSTORB, FLLASTORB, FLLEN
          FROM FILE, CAT
1)        WHERE FILE.TAPEID=CAT.TAPEID AND
2)        [CAT.FILENUM=FILE.FILENUM OR CAT.FILENUM=NULL] AND
3)        FILE.NUMITEMS=0 AND
4)        CAT=<'OZONE'> AND
5)        [FLSTART<='------------' AND FLSTOP>='------------'] AND
6)        FILE.TAPEID=<'DPFL0001', 'DPFL0002', ......, 'DPFL0016'>;
```

For each time range the query was submitted four times. In the first job, the query appeared as is shown above. In the second, the CAT=<'OZONE'> phrase was moved to the end. In the third, the table joins, phrases 1 and 2 , were moved to the end, and in the final test, the tape qualifier phrase, 6 , was moved to the beginning.

Tables giving the results of the four variations for time range 1 and 2, respectively, are given below. The results include the statistics for the job itself and the detached process, as reported in the VAX account file.

ORACLE QUERY RESULTS - 2,796 Rows Retrieved

Where Clause Ordering	Average Response Time (Sec)	% Degradation over Best Results	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's
1,2,3,4,5,6	.177	+ 1.7	494.	392.57	4,260.
1,2,3,5,6,4	.183	+ 5.2	511.	432.38	1,766.
3,4,5,6,1,2	.174	---	487.	381.72	4,260.
6,1,2,3,4,5	.181	+ 4.0	506.	430.52	1,772.

ORACLE QUERY RESULTS - 1,412 Rows Retrieved

Where Clause Ordering	Average Response Time (Sec)	% Degradation over Best Results	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's
1,2,3,4,5,6	.320	+ 1.9	452.	354.77	4,261.
1,2,3,5,6,4	.330	+ 5.1	466.	393.79	1,771.
3,4,5,6,1,2	.314	---	444.	344.40	4,261.
6,1,2,3,4,5	.335	+ 6.7	473.	402.45	1,771.

There is about a 5-6% difference between the best and worst connect time in each table. While the algorithm used by the ORACLE DBMS for parsing the query is not known, it is important to no[illegible]rations in query response time do result from reorganizing the where clause. A more detailed discussion of parsing logic can be found in the INGRES Version 1.4 testing which follows.

It should be noted here that in each table, the total CPU time and total number of direct I/O's are much lower for the first and third entries than the second and fourth. In the first and third entries, phrase 4 (CAT=<'OZONE'>) appears before phrase 6 (FILE.TAPEID=<'DPFL0001',..., 'DPFL0016'>). In the other two entries, phrase 6 appears before phrase 4. Two statements may be made about this. First, the CAT table has far fewer rows than the FILE table. Second, the "=" is on a single value in phrase 4 and is on a group of 16 values in phrase 6. It seems plausible that ORACLE might prefer to select on phrase 4 before phrase 6.

INGRES Version 1.4 Results

A test was run to determine whether the order of the various parts of the where clause in a complex query had an impact on the rate of retrieval. These tests employed the newer INGRES Version 1.4.

For this test, a PCDB application was used, where the data base contained 13,631 records stored in five tables (see Appendix II). The query that was executed follows, with the numbers to the left of the query signifying the various predicates in the where clause.

```
          RANGE OF F IS FILE, C IS CAT
          RETRIEVE (C.CAT, C.CATEGORY, C.FUNCTION, F.FILENUM, F.FLSTART,
                   F.FLSTOP, F.FLFIRSTORB, F.FLLASTORB, F.FLLEN) WHERE
1)                 F.TAPEID=C.TAPEID AND
2)                 (C.FILENUM=F.FILENUM OR C.FILENUM=0) AND
3)                 F.NUMITEMS=0 AND
4)                 C.CAT='OZONE' AND
5)                 (F.FLSTART<='710401000000'† AND F.FLSTOP>='700801000000') AND
6)                 F.TAPEID='DPFL*'
```

†Using this time range, 2,796 rows were retrieved. A second set of runs was made using '710201000000' and '701001000000' for FLSTART and FLSTOP, respectively. Using this time range, 1,412 rows were retrieved.

In the two tables below, the results are shown from querying the PCDB data base where the ordering of the predicates in the where clause was varied. The first table shows the results using the first set of values for FLSTART and FLSTOP, and the second table reports the results using the second, more restrictive, set of values for FLSTART and FLSTOP. In each table, the results of four queries are reported using statistics from the VAX accounting file. In the first, the where clause was ordered as shown in the query above. In the second, the C.CAT='OZONE' clause was moved to the end of the where clause. In the third, clauses one and two (F.TAPEID=C.TAPEID and (C.FILENUM=F.FILENUM OR C.FILENUM=0)) were moved to the end of the where clause. In the final query, the match for tapeid (F.TAPEID="DPFL*') was inserted as the first part of the where clause. In all queries, the FILE table was defined as 'ISAM' on TAPEID and FILENUM and the CAT table was defined as 'hash' on TAPEID, FILENUM, ITEM, and CAT. There were no secondary indices created.

INGRES QUERY RESULTS

Ordering of Where Clause	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)
1,2,3,4,5,6	.054	---	151.	94.19
1,2,3,5,6,4	.053	- 1.9	149.	93.68
3,4,5,6,1,2	.053	- 1.9	149.	92.74
6,1,2,3,4,5	.053	- 1.9	147.	90.82

INGRES QUERY RESULTS

Ordering of Where Clause	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)
1,2,3,4,5,6	.066	---	93.	57.30
1,2,3,5,6,4	.064	- 3.0	90.	54.52
3,4,5,6,1,2	.063	- 4.5	89.	54.15
6,1,2,3,4,5	.062	- 6.1	88.	52.74

The difference in connect time and CPU time which appears with the same pattern between the two tables suggests that the query optimization that takes place on an INGRES query is somewhat dependent on the structure of the where clause. A few examples of how where clause structure might affect performance are given here.

When a complex where clause is specified that performs a join on two (or more) tables, the DBMS must choose one of the tables to begin selection on. Some possibilities of criteria for this selection might be table sizes or the number of pieces of the where clause involved with each table. If the DBMS does not keep statistics about each table (e.g., table size), the selection may be a random one.

Once a table has been chosen to begin selection on, the where clause is parsed for those pieces of the where clause concerning the selected table. The DBMS optimization algorithm specifies a general ordering for solving pieces of the where clause. That is, the first thing the parser might look for is a piece of the where clause specifying the value of the primary key. Next, it might look for a piece specifying the value of a secondary key. After that it might look for a piece where an '=' is specified on a non-indexed field, followed by a clause where a '>' or '<' is specified. A simple example of how performance might be affected by this follows. Suppose the query reads:

```
RETRIEVE (A.ALL)
  WHERE A.SEX='MALE'
  AND A.COLOR='BLUE'
```

The field A.SEX can take on one of two values, 'MALE' or 'FEMALE', where the field of A.COLOR can take on many values. Therefore, in a given table, there are probably more duplicates in the SEX field than in the COLOR field. If the parser reads the first qualification and satisfies it first, in a table of 100 rows, 50 rows may be retrieved. Then, in applying the second qualification, if there are 10 colors, 5 rows may be retrieved. If, however, the parser reads and remembers the last qualification first, 10 rows may be retrieved by COLOR, followed by 5 rows retrieved by SEX.

The point to be made is that query optimization does exist. While the test conducted in this section did not show a wide variation in query performance due to where clause restructuring, the user should be aware of the fact that variation does exist. A good knowledge of the data in the data base and of the types of information most frequently requested from the data base can help the data base designer determine which fields to index. Testing variations in the where clause structuring may aid in improving data base query performance.

4.5 Effect of Ordering the Output from a Query

It is sometimes desirable to retrieve the results of a query in a predetermined order. This ordering of results may, however, affect the response time of the query. It is not uncommon for the DBMS to employ temporary tables to use for sorting the responses to a query.

A test was conducted to determine the extent of degradation in response time for various "sort" fields. Testing was conducted using the ORACLE 2.3 and INGRES 1.4 DBMSs.

ORACLE Version 2.3.2 Results

Using the PCDB application with 13,631 input records stored in the data base in 5 tables, a test was conducted to determine the impact of an 'ORDER BY' clause in the response time of a complex query.

The query which was executed was:

```
SELECT CAT, FORMAT, CATEGORY, FUNCTION, TAPE.TAPEID, FILE.FILENUM,
    FLSTART, FLSTOP, FLFIRSTORB, FLLASTORB, FLLEN
FROM TAPE, FILE, CAT
WHERE TAPE.TAPEID=CAT.TAPEID AND CAT.TAPEID=FILE.TAPEID AND
    [CAT.FILENUM=FILE.FILENUM OR CAT.FILENUM=NULL] AND
    FILE.NUMITEMS=0 AND
    [FLSTART<='900000000000' AND FLSTOP>='700000000000'] AND
    TAPE.TAPEID=
SELECT TAPEID FROM TAPE WHERE FORMAT='DPFL' AND
    [TPSTART<='900000000000' AND TPSTOP>='700000000000'];
```

All results are shown below and include both the job and detached process, as reported in the VAX account log.

ORACLE QUERY RESULTS - 5,185 Rows Retrieved

ORDER BY	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
No ORDER BY	.058	---	300.	240.76	1,814.	20,997.	2,062.
CAT, FLSTART	.138	+ 137.9	718.	478.69	5,394.	21,745.	2,077.
CAT, FUNCTION, FLSTART	.140	+ 141.4	724.	486.02	5,399.	21,773.	2,371.
CATEGORY, FUNCTION, FLSTART	.148	+ 155.2	766.	513.30	5,689.	21,836.	2,371.
CAT, CATEGORY, FUNCTION, FLSTART	.151	+ 160.3	783.	525.26	5,704.	21,863.	2,371.

The presence of an 'ORDER BY' clause more than doubled the query response time. Any time a sort is invoked, the CPU time and the number of I/O operations increases because of the shuffling of rows. As the 'ORDER BY' clause becomes more complex, the response time increases even more. It should be noted that in the test run here, the CATEGORY field is CHARACTER*30, the FUNCTION field is CHARACTER*50, CAT is CHARACTER*5, and FLSTART is CHARACTER*12.

INGRES Version 1.4 Results

Another important feature of querying that was tested was the effect of a specified ordering in the rows that were retrieved by a query. The query used in this testing was identical to one variation of the query used in the previous testing (Predicate Reordering) and is shown below. Also, as in the previous testing, INGRES Version 1.4 was used.

```
RANGE OF F IS FILE, C IS CAT
RETRIEVE (C.CAT, C.CATEGORY, C.FUNCTION, F.FILENUM, F.FLSTART, F.FLSTOP,
         F.FLFIRSTORB, F.FLLASTORB, F.FLLEN) WHERE
         F.TAPEID='DPFL*' AND
         F.TAPEID=C.TAPEID AND
         (C.FILENUM=F.FILENUM OR C.FILENUM=0) AND
         F.NUMITEMS=0 AND
         C.CAT='CZONE' AND
         (F.FLSTART<='710401000000' AND F.FLSTOP>='700801000000')
```

In this testing, as in the previous testing, a second set of times (FLSTART<='710201000000' AND FLSTOP>='701001000000') were also tested. The test was run once for each set of times with no ordering of the output and once with an ordering by CAT, CATEGORY, and FLSTART. When more than one column is specified for sorting, the results are sorted first by the leftmost column specified (i.e., CAT), then by CATEGORY and finally by FLSTART. In this testing there was no structure on the data base tables (i.e., 'heap'). It was thought that this would give a truer representation of the effect of the 'SORT BY' clause than if the tables were structured. The table which follows reports the results of the first and second set of times, respectively, from the VAX account log.

INGRES QUERY RESULTS

Time	SORT BY	Average Response Time (Sec)	% Degradation in Average Response Time	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
1	No SORT BY	.060	---	169.	106.01	2,773	4,588	1,234
	CAT, CATEGORY, FLSTART	.113	+ 88.3	317.	175.04	6,314	5,806	1,650
2	No SORT BY	.079	---	112.	67.18	2,022	2,401	1,230
	Sort by CAT, CATEGORY, FLSTART	.128	+ 62.0	181.	97.77	3,655	3,011	1,555

The addition of the 'SORT BY' clause has a severe impact on query performance. This is to be expected, especially for retrieval of large numbers of rows where the sort key is more than one field. INGRES accomplishes the sort by use of temporary tables created during the run and deleted at the end of the run. When a sort key is introduced, the number of direct I/O's, as well as buffered I/O's and page faults, is increased, as can be seen in the table.

4.6 Nested Queries

The purpose of the test which was performed here was to assess the impact of query nesting. In this test, a query was created which employed two levels of nesting and three table joins. In a control run, the two levels were separated into three queries with the results of one being used in the next. This eliminated the need for "joins" which were inherent in the nested query. This test was conducted using ORACLE 2.3 only.

ORACLE Version 2.3.2 Results

A test was designed to determine the impact of nested queries on response time. The PCDB application was used in the testing. The data base was first loaded with 13,631 input records into five tables. Then, several queries were executed.

The control run queries were structured as follows:

```
1)  SELECT TAPEID, TPSTART, TPSTOP FROM TAPE
      WHERE FORMAT = 'DPFL'
      AND TPSTART<= '99'
      AND TPSTOP>= '00';

2)  SELECT CAT, CATEGORY, FUNCTION, FILENUM FROM CAT
      WHERE TAPEID = 'DPFL0009'
      AND CAT = <'OZONE'>
      AND ITEM = NULL;

3)  SELECT FLSTART, FLSTOP, FLFIRSTORB, FLLASTORB, FLLEN, FILENUM
      FROM FILE
      WHERE TAPEID = 'DPFL0009'
      AND FLSTART<='99'
      AND FLSTOP>='00';
```

The nested query was structured as:

```
SELECT CAT, CATEGORY, FUNCTION, TAPE.TAPEID, FILE.FILENUM,
    FLSTART, FLSTOP, FLFIRSTORB, FLLASTORB, FLLEN
FROM TAPE, FILE, CAT
WHERE TAPE.TAPEID = CAT.TAPEID AND
    TAPE.TAPEID = FILE.TAPEID AND
    [CAT.FILENUM=FILE.FILENUM OR CAT.FILENUM=NULL] AND
    FILE.NUMITEMS = 0 AND
    [FLSTART<='701226' AND FLSTOP>='701225'] AND
    TAPE.TAPEID=
SELECT TAPEID FROM TAPE
    WHERE FORMAT = 'DPFL'
    AND [TPSTART<='701226' AND TPSTOP>='701225'];
```

The results of the two tests show that even though a 3 way join was necessary in the two-level nested query, it was still faster than running three queries. The results are shown here for both the jobs and detached processes (as reported in the VAX account file) and should be examined closely because the total statistics are somewhat misleading.

ORACLE QUERY RESULTS

Type of Query	Total Connect Time (Sec)	Total CPU Time (Sec)	Total Direct I/O's	Total Buffered I/O's	Total Page Faults
2 Level Nested Query	Job 35. DP* 29.	2.57 18.51	32 249	149 141	463 1,549
3 Separate Queries	Job 51. DP* 26.	6.55 14.16	36 190	903 789	471 976

* Indicates Detached Process

First, it should be noted that when the two level nested query was executed, 13 rows were returned. When the three separate queries were executed, 16, 1, and 363 rows were retrieved, respectively, for a total of 380 rows.

Looking at the results of the detached processes (labelled DP above), which perform the actual ORACLE tasks, the table joins had an obvious impact on performance. Thirteen rows were retrieved using 18.51 seconds of CPU time and 29 seconds of connect time. In the run where the query was broken down so that no joins were necessary, 380 rows were retrieved using only 14.16 seconds of CPU time and 26 seconds of connect time. However, the overhead of running three separate jobs, thus executing the non-ORACLE related tasks three times instead of one, was the contributing factor in the overall performance degradation of that run.

APPENDIX I

PACKET MANAGEMENT SYSTEM (PMS)
SCHEMAS FOR DATA BASE TESTING

PACKET MANAGEMENT SYSTEM

The Packet Management System (PMS) is partially intended to demonstrate the ability to use data base technology to provide various NASA end users with an improved capability to access data stored centrally in a packetized manner. The formal design of the data base schema for PMS had not been determined prior to the performance testing this document describes. Rather it was intended that results of this testing would be used as input for some of the decisions required during the formulation of the PMS schema design. Some preliminary results have, in fact, been responsible for the alteration of initial designs that were proposed prior to the start of the performance testing. What has evolved is a marriage of goals to the end that generic data base testing has proceeded which has often employed PMS-like data base applications to provide specific feedback to the PMS project to aid in system design considerations. The term "PMS-like" is used throughout the report because the data used in the testing simulated, but was not, actual PMS data.

The PMS-like data base designs used during the testing reflect changes made by the PMS project staff to the preliminary PMS data base designs. The designs applied in Section 2 were primarily based on alterations made to the originally proposed design. Some of the supplemental testing described in Section 4 employed the original design of the PMS. It was these test results that prompted the first major design revision to the proposed data base structure. The PMS data base requirements included the need to store and manage headers that prefixed information contained in packets of data. The header information to be managed was initially defined in the table that follows:

PMS HEADER INFORMATION

Field	Description	Length
SID	Source Identifier	8 bits
MID	Mission Identifier	8 bits
SSC	Source Sequence Count	16 bits
PL	Packet Length	8 bits
SDF	Source Data Format	8 bits
SHID	Secondary Header Identifier	8 bits
SIEC	Source ID Error Control	8 bits
TIME	Time	32 bits
PLI	Alternative Packet Length	32 bits
	TOTAL	128 bits

In addition, the next 384 bits of the packet were to be included as a single field (resulting in the DBMS storing the first 512 packet bits). Also, to facilitate PMS users, a 32 bit field for their UICs and a 63 byte comment field were to be managed in the data base as well. As a requirement, each packet would be identifiable through a unique value derived through the concatenation of the following fields: SID, MID, SSC, SDF. SIEC and TIME. These fields, when their values are concatenated together, constitute a primary key that is associated with only one header in the data base.

The basic ORACLE schema employed for this initial specification of PMS data base needs consisted of a single relation or table. The table had the following makeup:

ORACLE HEADER TABLE

Column Name	Description		Source
KEY	CHAR(10)	NONULL	Concatenation of 6 fields
SID	NUMBER	NONULL	Packet
MID	NUMBER	NONULL	Packet
SSC	NUMBER	NONULL	Packet
PLP	NUMBER	NONULL	Packet
SDF	NUMBER	NONULL	Packet
SHID	NUMBER	NONULL	Packet
SIEC	NUMBER	NONULL	Packet
TIME	NUMBER	NONULL	Packet
PLS	NUMBER	NONULL	PLI Field From Packet
SECHDR	CHAR(48)	NONULL	Next 384 bits in Packet
UIC	NUMBER		User ID Code (Optional)
COMMENT	CHAR(63)		User Comment (Optional)

This table formed the baseline from which alternative designs were made for the supplemental testing of ORACLE Versions 2.3.1 and 2.3.2 summarized in Section 4 and some of the Level 2 testing described for ORACLE in Section 3.

The fields that were chosen for imaging (indexing) varied depending on the test goals, although the KEY field was always imaged since it was a primary key and ORACLE 2.3 required that at least one field be indexed in a table.

The initial design for SEED appears pictorially in Figure I.1 and is summarized in Table I-1. In the figure, the upper portion of each block identifies the area name on the left and the calc field or via set on the right. The lower portion of each box is the record name. The R6_PKEY record contains the various PMS header fields required to be managed. The R7_COMMENT record is optionally present for a given P6_PKEY record. The R1_MID, R2_TIME, R3_IID (SDF), R4_SID and R5_UIC records provide direct access to header records through the designation of a value for a particular owner and the fetching of the corresponding R6_PKEY member records. As with ORACLE, this baseline schema was altered to test various designs. During this phase of testing, SEED version B.11.9 was used which did not have the Pointer Array indexing capability, so owner-member sets were the only means for direct access.

After test results had been obtained and reviewed using variations of the above schemas, the PMS project staff proposed a revision to the PMS data base design which was adopted for the generic performance testing. In order to increase ingestion rates, the number of fields was reduced and, because of project changes, the primary key was redefined. The fields defined to be managed under the revised PMS needs were:

PMS HEADER INFORMATION

Field	Description	Length
MID_SID	A concatenation of Mission and Source ID's	16 bits
TIME	Time	32 bits
SDF	Source Data Format	16 bits

In addition, the entire first 512 bits were required to be treated as a single field which could be broken apart by PMS software, a primary key consisting of the MID, SID, TIME, SSC and SDF fields (no SIEC) would be needed to uniquely identify a packet, and UIC and MESSAGE (replaced COMMENT) fields were necessary for end users to identify their packets.

SEED SCHEMA REPRESENTATION

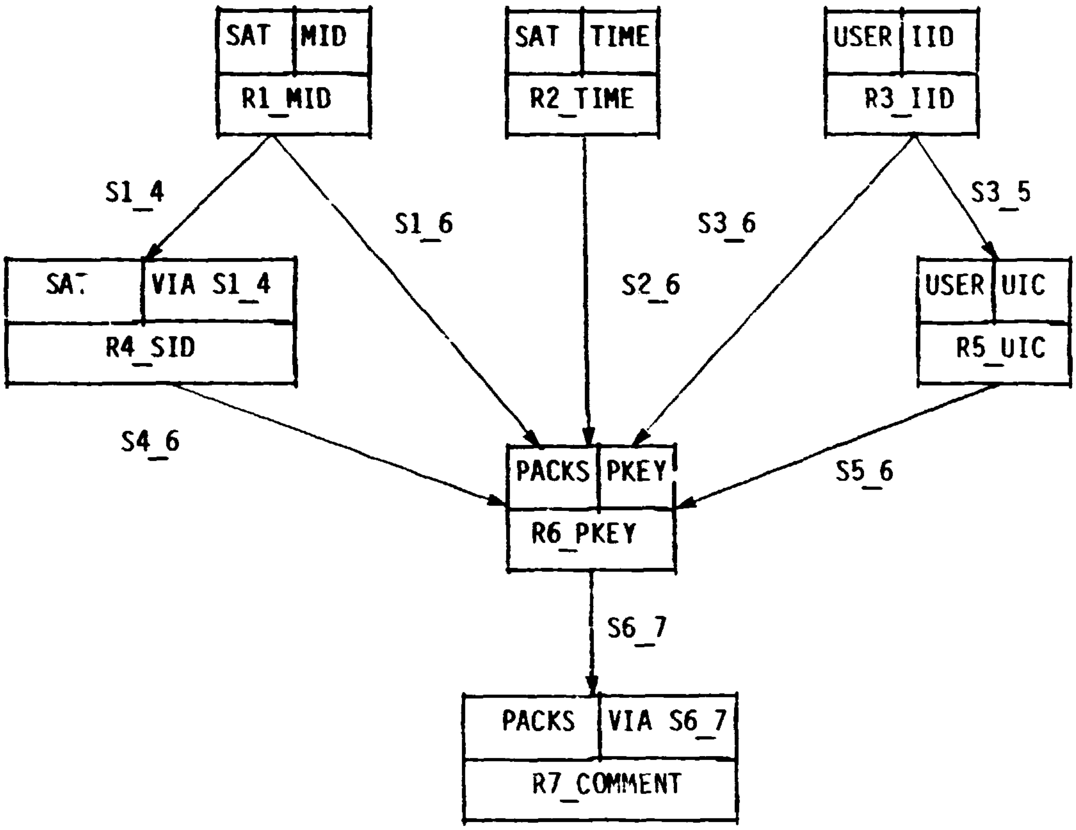


Figure I.1

SEED SCHEMA REPRESENTATION

Record Name	Field Name	Type
R1_MID	MID	INTEGER*2
R2_TIME	TIME	INTEGER*4
R3_IID	IID	INTEGER*2
R4_SID	SID	INTEGER*2
R5_UIC	UIC	INTEGER*4
R6_PKEY	PKEY	CHARACTER*10
	MIDP	INTEGER*2
	SIDP	INTEGER*2
	SSC	INTEGER*2
	SDF	INTEGER*2
	SIEC	INTEGER*2
	TIMEP	INTEGER*4
	UICP	INTEGER*4
	PL	INTEGER*2
	SHID	INTEGER*2
	PLI	INTEGER*4
	BITS384	CHARACTER*48
R7_COMMENT	COMMENT	CHARACTER*63

Table I-1

The ORACLE schema which subsequently evolved to accommodate the proposed changes is presented in the specification below:

REVISED ORACLE HEADER TABLE

Column Name	Description		Source
PRIMARY_KEY	CHAR(9)	NONULL	Concatenation of 5 fields
MID_SID	NUMBER	NONULL	Concatenation of MID and SID
TIME	NUMBER	NONULL	Packet
UIC	NUMBER		User ID Code (Optional)
SDF	NUMBER	NONULL	Packet
MESSAGE	CHAR(31)		User Message (Optional)
HEADER	CHAR(64)	NONULL	First 512 bits of packet

This design was employed for much of the Level 1 ORACLE testing described in Section 2. For most of that testing, PRIMARY_KEY, MID_SID and TIME were imaged (indexed) fields. The UIC and COMMENT fields were predominently employed as the variable elements of the tests summarized in Section 2. This was also true for the same testing done with SEED and INGRES. It should be noted that originally ORACLE did not permit the specification of an index or image for multiple fields unless those fields were concatenated and stored in the data base in that form. This was why the PRIMARY_KEY had to be present as a separate field and MID and SID had to be concatenated. Not until a release of ORACLE Version 3 would such a capability become available.

The revised SEED schema is summarized in Figure I.2 and Table I-2. SEED Version C.0 was received in time to be used in the testing with this baseline schema. It offered the Pointer Array feature (a B-tree direct access implementation) which was implemented during some of the testing discussed in Section 2.

REVISED SEED SCHEMA REPRESENTATION

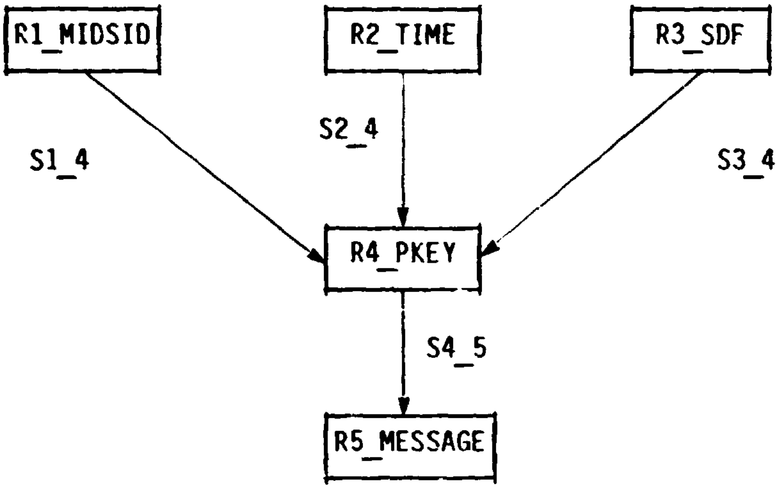


Figure I.2

REVISED SEED SCHEMA REPRESENTATION

Record Name	Field Name	Type
R1_MIDSID	MIDSID	INTEGER*2
R2_TIME	TIME	INTEGER*4
R3_SDF	SDF	INTEGER*2
R4_PKEY	PKEY	CHARACTER*9
	MIDSID	INTEGER*2
	TIME	INTEGER*4
	SDF	INTEGER*2
	HEADER	CHARACTER*64
	UIC	INTEGER*4
R5_MESSAGE	MESSAGE	CHARACTER*31

Table I-2

Testing began on the INGRES data base management system at this time. Because INGRES employs the use of multifield keys, the following table structure was derived.

INGRES HEADER TABLE

Column Name	Description	Source
MID_SID	I*2	Concatenation of MID and SID
SSC	I*2	Packet
SDF	I*2	Packet
TIME	I*4	Packet
UIC	I*2	User ID Code (Optional)
MESSAGE	CHAR(31)	User Message (Optional)
HEADER	CHAR(64)	First 512 bits of packet

The combination of MID_SID, SSC, SDF, and TIME formed the multifield key used to uniquely identify each packet.

APPENDIX II

PILOT CLIMATE DATA BASE MANAGEMENT SYSTEM (PCDBMS)

SCHEMAS FOR DATA BASE TESTING

PILOT CLIMATE DATA BASE MANAGEMENT SYSTEM

The Pilot Climate Data Base (PCDB) was developed as a means of providing on-line catalog, tape inventory, and data access capabilities to a portion of the scientific community. An initial design was developed and implemented on the VAX 11/780. Subsequently, because of greater variance between tapes (i.e., the type of information present on them), and because of feedback on the "prototype" design, it was determined that a design change was necessary.

Most of the information presented in this document was obtained from testing performed using the original design. In that design, there were four levels of information - a tape level, file level, item level, and cat level. As the information was read from the tape, the first byte of each record identified the record as tape, file, item or cat level information. A description of the data present in each of the four levels of information is shown in the tables below.

TAPE LEVEL

Field	Description	Length (Bytes)
RECORDID	Record identification	1
TAPEID	Tape ID	15
MISSION	Mission name for this satellite	15
SENSOR	Sensor equipment name	10
FORMAT	Data Format	10
PROJNUM	Project sequence number	15
GENDATE	Date tape was originally generated	13
INVDATE	Date tape was inventoried	9
ARCHIVER	Name of person inventorying tape	12
LOCAT	Location of tape	15
NUMFILES	Number of files on the tape	8
TPFIRSTORB	Minimum orbit number on tape	8
TPLASTORB	Maximum orbit number on tape	8
TPSTART	Start time on tape	13
TPSTOP	Stop time on tape	13
TPALGORITHM	Processing algorithm used	5
COORDSYS	Map coordinate system	15
SYNOPSTART	Synoptic start time	13
SYNOPSTOP	Synoptic stop time	13
INCORB	Orbital inclination	8
ORBPER	Orbital period	8
MSCANG	Maximum scan angle	8
SCNINC	Scan angle increment	8
NODPRT	Node procession rate	8
		TOTAL 251

FILE LEVEL

Field	Description	Length (Bytes)
RECORDID	Record identification	1
FILENUM	Sequential file number	8
FLFIRSTORB	First orbit in file	8
FLLASTORB	Last orbit in file	8
FLSTART	Start time of file	13
FLSTOP	Stop time of file	13
FLALGORITHM	Processing algorithm used	5
NUMITEMS	Number of items in file	8
FLLEN	Size of file (bytes)	8
EQXTIM	Equator crossing time	13
EQXLNG	Equator crossing longitude	8
		TOTAL 93

ITEM LEVEL

Field	Description	Length (Bytes)
RECORDID	Record identification	1
ITSTART	Start time of item	13
ITSTOP	Stop time of item	13
ITEM	Mnemonic of item	5
NAME	Description of item	30
RECNUM	Physical block number of record	8
ITALGORITHM	Processing algorithm used	5
ITLEN	Maximum size of item (bytes)	6
		TOTAL 81

CAT LEVEL

Field	Description	Length (Bytes)
RECORDID	Record identification	1
CAT	Climate parameter mnemonic	5
CATEGORY	Description of cat	30
FUNCTION	Method to drive cat	50
		TOTAL 86

The ORACLE data base that was employed to manage the pilot climate data consisted of five tables - a tape table, file table, item table, item description table, and cat table. The rows in the tape table were uniquely defined by the tapeid. This tapeid was placed in the file table also, so that the rows in the file table were uniquely defined by a combination of tapeid and file number. Likewise, these were placed in the item table to uniquely identify those rows by tapeid, file number, and item. Finally, the tapeid and file number were placed in the cat table, so that in combination with the cat field, rows could be uniquely defined. The ITEM field was not included here because it could be a null field. A fifth table, the item description table, contained all of the items and their names. By placing these in a separate table, it was not necessary to store redundant data in the item table.

Each table is shown below. Imaged fields are denoted by an asterisk following the column name.

TAPE TABLE

Column Name	Description	
TAPEID*	CHAR(15)	NONULL
MISSION*	CHAR(15)	NONULL
SENSOR*	CHAR(10)	NONULL
FORMAT*	CHAR(10)	NONULL
PROJNUM	CHAR(15)	
GENDATE	CHAR(12)	
INVDATE	CHAR(8)	NONULL
ARCHIVER	CHAR(12)	
LOCAT	CHAR(15)	
NUMFILES	NUMBER	
TPFIRSTORB*	NUMBER	
TPLASTORB*	NUMBER	
TPSTART*	CHAR(12)	
TPSTOP*	CHAR(12)	
TPALGORITHM	CHAR(5)	
COORDSYS	CHAR(15)	
SYNOPSTART	CHAR(12)	
SYNOPSTOP	CHAR(12)	
INCORB	NUMBER	
ORBPER	NUMBER	
MSCANG	NUMBER	
SCNINC	NUMBER	
NODPRT	NUMBER	

FILE TABLE

Column Name	Description	
TAPEID*	CHAR(15)	NONULL
FILENUM*	NUMBER	NONULL
FLFIRSTORB	NUMBER	
FLLASTORB	NUMBER	
FLSTART*	CHAR(12)	
FLSTOP*	CHAR(12)	
FLALGORITHM	CHAR(5)	
NUMITEMS	NUMBER	
FLLEN	NUMBER	
EQXTIM	CHAR(12)	
EQXLNG	NUMBER	

ITEM TABLE

Column Name	Description	
TAPEID*	CHAR(15)	NONULL
FILENUM*	NUMBER	NONULL
ITEM*	CHAR(5)	NONULL
ITSTART	CHAR(12)	
ITSTOP	CHAR(12)	
RECNUM	NUMBER	
ITALGORITHM	CHAR(5)	
ITLEN	NUMBER	

CAT TABLE

Column Name	Description	
TAPEID*	CHAR(15)	NONULL
FILENUM*	NUMBER	
ITEM	CHAR(5)	
CAT*	CHAR(5)	NONULL
FUNCTION	CHAR(50)	
CATEGORY	CHAR(30)	
RECNUM	NUMBER	

ITEM_DESCR TABLE

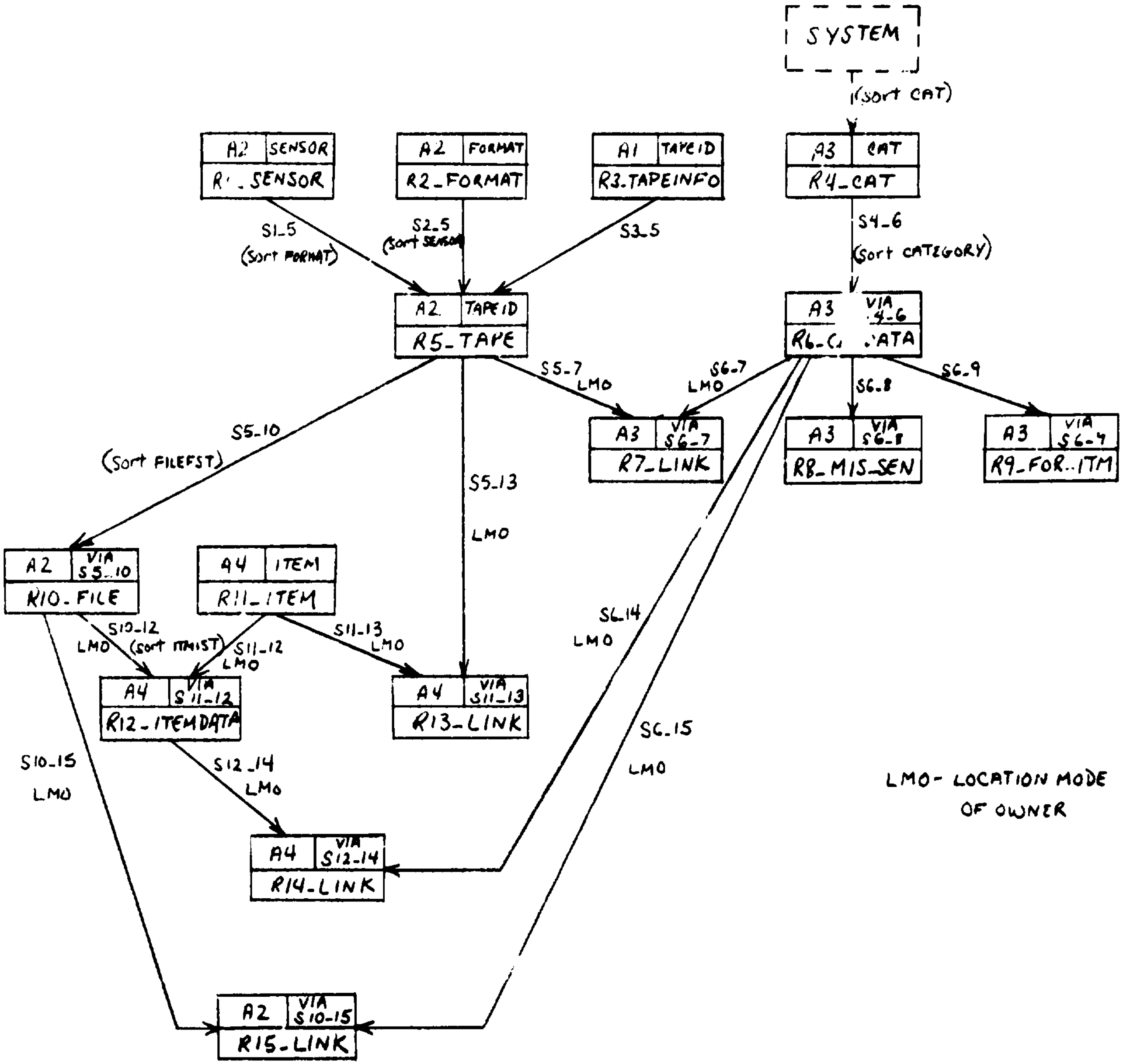
Column Name	Description	
ITEM*	CHAR(5)	NONULL
NAME	CHAR(30)	

The design used in SEED is shown in Figure II.1. Each block in the diagram is described as follows.

<table>
<tr><td>Area Name</td><td>Calc Key
or Via Set</td></tr>
<tr><td colspan="2">Record Name</td></tr>
</table>

Each set name is shown by the line connecting the owner and member of the set. Any other important set information (i.e., sort field and/or set occurrence selection through location mode of owner) is shown also. The design was developed with a specific set of queries in mind. Because of this, there is a certain amount of redundant data in the data base. The main core of data is located in the following records:

SEED SCHEMA REPRESENTATION



LMO- LOCATION MODE
OF OWNER

Figure II.1

- R3_TAPEINFO contains the tape level information that is not likely to be queried on . Examples of data in this record are COORDSYS, SYNOPSTART, SYNOPSTOP, and ARCHIVER.

- R5_TAPE contains the tape level information of importance in querying, i.e., TAPEID, MISSION, SENSOR, etc.

- R10_FILE contains all file level information.

- R11_ITEM contains the item mnemonic and description.

- R12_ITEMDATA contains the actual item information, i.e., item start and stop times, item length, etc.

- R4_CAT contains the climate parameter mnemonic.

- R6_CATDATA contains the cat information, category and function.

The remaining records are either redundant data included for speed of retrieval of information (R1_SENSOR, R2_FORMAT, for example) or are "link" records linking occurrences of 2 records together (R7_LINK links an occurrence of a tape record with an occurrence of cat level information).

The INGRES data base consisted of the same five tables as the ORACLE data base. Because column types are defined differently in ORACLE and INGRES, the tables, showing column name and type, are given below. In each table, the multifield keys are identified with asterisks.

TAPE TABLE

Column Name	Description
TAPEID*	CHAR(15)
MISSION	CHAR(15)
SENSOR	CHAR(10)
FORMAT	CHAR(10)
PROJNUM	CHAR(15)
GENDATE	CHAR(12)
INVDATE	CHAR(8)
ARCHIVER	CHAR(12)
LOCAT	CHAR(15)
NUMFILES	I*4
TPFIRSTORB	I*4
TPLASTORB	I*4
TPSTART	CHAR(12)
TPSTOP	CHAR(12)
TPALGORITHM	CHAR(5)
COORDSYS	CHAR(15)
SYNOPSTART	CHAR(12)
SYNOPSTOP	CHAR(12)
INCORB	I*4
ORBPER	I*4
MSCANG	I*4
SCNINC	I*4
NODPRT	I*4

FILE TABLE

Column Name	Description
TAPEID*	CHAR(15)
FILENUM*	I*4
FLFIRSTORB	I*4
FLLASTORB	I*4
FLSTART	CHAR(12)
FLSTOP	CHAR(12)
FLALGORITHM	CHAR(5)
NUMITEMS	I*4
FLLEN	I*4
EQXTIM	CHAR(12)
EQXLNG	I*4

ITEM TABLE

Column Name	Description
TAPEID*	CHAR(15)
FILENUM*	I*4
ITEM*	CHAR(5)
ITSTART	CHAR(12)
ITSTOP	CHAR(12)
RECNUM	I*4
ITALGORITHM	CHAR(5)
ITLEN	I*4

CAT TABLE

Column Name	Description
TAPEID*	CHAR(15)
FILENUM*	I*4
ITEM*	CHAR(5)
CAT*	CHAR(5)
FUNCTION	CHAR(50)
CATEGORY	CHAR(30)
RECNUM	I*4

ITEM_DESCR TABLE

Column Name	Description
ITEM*	CHAR(5)
NAME	CHAR(30)

APPENDIX III
SEED QUERY DESCRIPTION

1.) Calc on PKEYs (5%)

A.) Read in a record from the same file used to load the data base

B.) Form the primary key, PKEY

C.) Using the OBTNC command, find and get the correct R6_PKEY record (OBTNC finds and gets a record with a calculated key - PKEY)

D.) Repeat A through C for 5% of the input records to the data base. Mean statistics given in query table are for one iteration of this loop.

2.) Find PKEYs through OBTNU (.5%)

A.) Read in a record from the same file used to load the data base, to get an MID, SID, SDF and TIME

B.) Set up the "using" list and the "relation" list which is necessary to use the OBTNU command

C.) Using the OBTNU command, find all of the R6_PKEY records with the correct MID, SID, SDF and TIME (OBTNU finds and gets a record which meets a user-specified set of conditions)

D.) Repeat A through C for .5% of the input records to the data base. Mean statistics given in query table are for one iteration of this loop.

3.) Calc MID, find SID, find PKEYs (.5%)

A.) Read in a record from the same file used to load the data base, to get an MID, SID, SDF and TIME

B.) Using the FINDC command, find the correct R1_MID record (FINDC finds a record with a calculated key - MID)

C.) Using the OBTNPO command on the set with R1_MID as owner and R4_SID as member, find and get R4_SID records until the correct one is found. (OBTNPO finds and gets a record positionally within a set)

D.) Using the OBTNPO command on the set with R4_SID as owner and R6_PKEY as member, find all of the R6_PKEY records with the correct SDF and TIME. (OBTNPO finds and gets a record positionally within a set)

E.) Repeat A through D for .5% of the input records to the data base. Mean statistics given in query table are for one iteration of this loop.

4.) Calc MID, find PKEYs (.5%)

A.) Read in a record from the same file used to load the data base, to get an MID, SID, SDF and TIME

B.) Using the FINDC command, find the correct R1_MID record (FINDC finds a record with a calculated key - MID)

C.) Using the OBTNPO command on the set with R1_MID as owner and R6_PKEY as member, find all of the R6_PKEY records with the correct SID, SDF and TIME (OBTNPO finds and gets a record positionally within a set)

D.) Repeat A through C for .5% of the input records to the data base. Mean statistics given in query table are for one iteration of this loop.

5.) Calc TIME, find PKEYs (.5%)

A.) Read in a record from the same file used to load the data base, to get an MID, SID, SDF and TIME

B.) Using the FINDC command, find the correct R2_TIME record (FINDC finds a record with a calculated key - TIME)

C.) Using the OBTNPO command on the set with R2_TIME as owner and R6_PKEY as member, find all of the R6_PKEY records with the correct MID, SID and SDF (OBTNPO finds and gets a record positionally within a set)

D.) Repeat A through C for .5% of the input records to the data base. Mean statistics given in query table are for one iteration of this loop.

6.) Calc SDF, find PKEYs (.5%)

A.) Read in a record from the same file used to load the data base, to get an MID, SID, SDF and TIME

B.) Using the FINDC command, find the correct R8_SDF1 record (FINDC finds a record with a calculated key - SDF1))

C.) Using the OBTNPO command on the set with R8_SDF1 as owner and R6_PKEY as member, find all of the R6_PKEY records with the correct MID, SID and TIME (OBTNPO finds and gets a record positionally within a set)

D.) Repeat A through C for .5% of the input records to the data base. Mean statistics given in query table are for one iteration of this loop.

7.) Find PKEYs sequentially (.5%)

A.) Read in a record from the same file used to load the data base, to get an MID, SID, SDF and TIME

B.) Using the OBTNAP command, search sequentially through the data base for all R6_PKEY records with the correct MID, SID, SDF and TIME (OBTNAP finds and gets a record by treating the entire data base as a sequential file)

C.) Repeat A and B for .5% of the input records to the data base. Mean statistics given in query table are for one iteration of this loop.

8.) Calc on MIDSID, find PKEYs (.5%)

A.) Read in a record from the same file used to load the data base, to get an MID, SID, SDF and TIME

B.) Form the combination MIDSID

C.) Using the FINDC command, find the correct R1_MIDSID record (FINDC finds a record with a calculated key - MIDSID)

D.) Using the OBTNPO command on the set with R1_MIDSID as owner and R6_PKEY as member, find all of the R6_PKEY records with the correct SDF and TIME (OBTNPO finds and gets a record positionally within a set)

E.) Repeat A through D for .5% of the input records to the data base. Mean statistics given in query table are for one iteration of this loop.

9.) OBTNI indexed value, on PKEYs (5%)

A.) Read in a record from the same file used to load the data base, to get an MID, SID, SDF and TIME

B.) Using the FINDI command on the indexed ,tem, find all the R6_PKEY records with the correct MID, SID, SDF and TIME (FINDI finds an "indexed" record)

C.) Repeat A and B for 5% of the input records to the data base. Statistics given in query table are for one iteration of this loop.

10.) Find MESSAGE sequentially (first 5%)

A.) Using the OBTNAP command, access the "FIRST" R5_MESSAGE record in the named area. (OBTNAP finds and gets a record by treating the entire data base as a sequential file)

B.) Repeat A above, accessing the "NEXT" R5_MESSAGE record each time in the named area for 5% of the R5_MESSAGE records. Mean statistics given in query table are for one iteration of this loop.

APPENDIX IV
POST-PUBLISHED ORACLE TEST RESULTS

This appendix is reserved for future ORACLE test results to be added to the document as they become available.

BIBLIOGRAPHIC DATA SHEET

1. Report No.
CR 170615

2. Government Accession No.

3. Recipient's Catalog No.

4. Title and Subtitle
RESULTS OF DATA BASE MANAGEMENT SYSTEM PARAMETERIZED PERFORMANCE TESTING RELATED TO GSFC SCIENTIFIC APPLICATIONS

5. Report Date
June 30, 1983

6. Performing Organization Code

7. Author(s) Carol H. Carchedi, Thomas L. Gough,
Herbert A. Huston

8. Performing Organization Report No.
BTS2-82-45/rd 1021

9. Performing Organization Name and Address
Business and Technological Systems, Inc.
Aerospace Building, Suite 440
10210 Greenbelt Road
Seabrook, Maryland 20706

10. Work Unit No.

11. Contract or Grant No.
NAS5-26728

13. Type of Report and Period Covered

12. Sponsoring Agency Name and Address
NASA/Goddard Space Flight Center
Code 931
Greenbelt, Maryland 20771

14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract
This document summarizes the results of a variety of tests designed to demonstrate and evaluate the performance of several commercially available data base management system (DBMS) products compatible with the Digital Equipment Corporation VAX 11/780 computer system. The tests were performed on the INGRES, ORACLE, and SEED DBMS products employing applications that were similar to scientific applications under development by NASA. The objectives of this testing included determining the strength and weaknesses of the candidate systems, performance trade-offs of various design alternatives and the impact of some installation and environmental (computer related) influences.

17. Key Words (Selected by Author(s))
Benchmark
VAX
DBMS
INGRES
ORACLE
SEED

18. Distribution Statement

19. Security Classif. (of this report)

20. Security Classif. (of this page)
U

21. No. of Pages
257

22. Price*

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