A Performance Analysis of Relational and Hierarchical Database Packages based on Query Response

by Warren G. Weis

Each year the M Technology Association sponsors a student research competition for which full-time students submit papers expressing original ideas and research efforts related to the M language. The Student Research Award is made possible by contributions to the Michael Distaso Memorial Fund. The MTA is honored to present this year's award to Warren G. Weis, a graduate student at the University of California, Davis.

Abstract

The Oracle, FileMan, and M/SQL commercial database packages and a database of the author's own devising are analyzed and compared for performance on data retrieval operations. This is done by setting up a database on each package and timing data retrieval operations. The implementation of this experiment is described and the results made available for eight timing runs on databases based on from 5,000 to 40,000 records increasing in increments of 5,000. Some tentative conclusions are drawn, and the author's own observations on factors he believes may have affected the results are discussed along with possible areas of improvement for each database package.

1. Introduction

This paper is based on a study supported by the Department of Veterans Affairs (VA). The purpose of this study is to compare the performance of VA FileMan version 20, a database package developed and maintained by the VA, with the performance of three other database packages: M/SQL, a MUMPS based database package developed by InterSystems; Oracle, probably the most prominent relational database package on the market today; and an ad-hoc pure MUMPS database developed by the author. It was decided early on to limit the scope of this investigation to a study of how well these products do on query response. That is, how quickly a data retrieval request is satisfied. As queries may well constitute up to 90% of the operations of a typical database, this seemed like a good criterion to judge the ability of a database package to satisfy the needs of its users. What follows is a summary of the initial results.

2. Evaluation Issues

The typical query can be broken down into three phases:

1.) The user supplies information necessary to perform the query.

2.) This information, assuming it is sufficient to retrieve data from the database, is transformed into a form which the database package can use. The appropriate searches and/or sorts are performed. The retrieved information is put into some sort of storage.

3.) The retrieved information is transformed into a form visible to the user.

This last step may be skipped if the purpose of the data retrieval is to perform a backup.

The next step was to determine what quality of a query to judge performance on. Should user friendliness be considered here, or the more obvious quality of quickness of response? It was decided that time would be used to judge performance. Performance, in terms of time needed to perform the operation, is easier to measure than such abstract qualities as user friendliness. Nevertheless, I mention this quality here as I do not believe it should be overlooked in the evaluation of database packages. The ease with which the average user can find the most effective means to implement his or her request must contribute, in an indirect way, to the overall performance of a database.

Another fundamental problem was which database model to base the performance analysis on. FileMan is a database package which was designed to implement databases based on the hierarchical model. Oracle is a package which seeks to implement as closely as possible the relational database model as conceived of by C. J. Date and E. F. Codd[1]. I wished to judge performance of the various packages as closely as possible and felt that comparing a hierarchical database with a relational one would be comparing apples to oranges: It would be too easy to set up the hierarchical database structure such that it optimized query tests to the disadvantage of the relational database. On the other hand, a strict hierarchical model does not have the flexibility of the rela-

tional model. If sample databases were to be implemented in strict hierarchical and relational form, then there might be queries which would prove either extremely difficult or impossible for the hierarchical database to satisfy. I decided to implement a relational model on all database packages and made an interesting discovery: Relational models can be implemented relatively easily on hierarchical packages.

The last consideration was to what degree each database would be optimized. The concern here was that this could very well turn into a contest of my ability to optimize one database package versus another. It seemed unlikely that this would prove to be equal. I therefore decided to do the absolute minimum necessary to set up each database. There were two reasons why this approach was of interest. First, it might come closer to the performance a naive user might expect to get from each database package. Second, it would be possible to see the absolute minimum in performance each database package had to offer. As an objective measurement I hoped this would prove relatively easy to achieve.

Having determined that the model to be implemented was to be relational and that time was to be used in judging performance, the question became which of the three phases mentioned above to measure. The speed with which phase 1 is completed is up to the user. The speed of phase 3 depends on how fast the operating system can do I/O. As only phase 2 is entirely in the hands of the database management system, I decided to time phase 2. Time was measured in terms of overall time needed for the query. That is, instead of using a breakdown of time into user, cpu and system measurements as is done with the UNIX Time command, actual real time was measured in seconds and microseconds using a SUN library function measuring microseconds and seconds in actual system time.

In order to make the analysis as impartial as possible all four databases were isolated on a single SPARC workstation. This workstation had 16 Megabytes of memory and a 2 Gigabyte hard drive. During timing runs no user activities are allowed on the workstation.

3. The Database Population Chosen

The backup of a large hospital database consisting of 400,000 records was made available by the IDX Corporation located in Vermont, which is conducting a similar study. This database was stored in the form of a large MUMPS global called ^PT (short for patient) using the InterSystems utility function ^%GO. Each record of a patient is stored by ID number. The ID number is used as the first subscript of the global. A second subscript is used as an identifier for a specific table in the record. Thus, all records for patient 100 can be found under ^PT(100). Data for a table of general patient information can be found at location ^PT(100,0). Data for patient 100's guarantor can be found at location ^PT(100,4). Each field of each table is separated by a single delimiter consisting of the character ^'. This delimiter is also the standard delimiter for FileMan tables which made transformation of the database to FileMan format easier. In summary, the format for table zero for patient 100 would look like this:

```
^PT(100,0)=A1^A2^A3^A4^A5...[2]
```

There are about thirty such tables. For the purposes of this study the five tables which appeared to be most densely populated were chosen. A brief of each table follows:

```
^PT(id, 0)=A1^A2^A3...
      Fields used:
      A1= Patient Name
      A2=Patient Social Security
      A3=Patient Sex
      ^PT(id,3)=A1^A2^A3...
      Fields used:
      A2=Activating Operator
      ^PT(id,4)=A1^A2^A3...
      Fields used:
      A1=First line of Guarantor's address.
      ^PT(id,5)=A1^A2^A3...
      Fields used:
      A1=Patient Employer
      ^PT(id,60)=A1^A2^A3...
      Fields used:
      A2=Is father a Veteran? (yes/no)
```

4. The Database Implementation On Oracle

As mentioned earlier, Oracle is probably the most widely used implementation of the relational database model on the market today. The relational model, as opposed to the hierarchical is based on essentially independent tables loosely linked to each other by data or keys which are common to the linked tables. Every row of every table must be uniquely identified by a primary key. Related tables are linked by foreign keys. When data from more than one table is being retrieved in a query, these tables are said to be joined on a field whose value is the same in each. In the database above, for example, each table is linked or joined by the patient ID. I could, of course, join tables on a different field if I wished to: For example, the Sex field in Guarantor and Patient if I wished to retrieve all references to male patients or guarantors in the database. The interested reader is referred to any number of excellent references for a more detailed description of this model.[3]

Oracle is a Relational Database Management System which runs on top of a host operating system. It has its own storage space allocated when the system is set up and its own caches

to speed up execution of frequently executed commands. Records may or may not be physically located near each other. If the system administrator requests data clustering then an effort is made to do so. Fields of a record may be indexed using a B or B+ tree structure.[4] The primary key of a table row is always indexed.[5]

The main driver of the Oracle RDBMS, SQLPLUS, uses the SQL query language to retrieve and modify data tables. In addition, in the last few years Oracle has introduced PL/SQL, a pseudo programming language strongly modeled after the programming language ADA to enable the database programmer to perform blocks of SQL statements. PL/SQL has the additional advantage that compiled versions of its SQL statements are stored rather than being interpreted at run time as is the case with command line SQL queries.

For the implementation of the ^PT database on Oracle, each patient record was split into flat files containing *only* rows for a single table. The patient ID was added onto the beginning of each table entry and serves as the unique identifier (primary key) for each row. Relational tables were then set up for the Oracle database corresponding to the entries contained in each flat file. See appendix A for the table definitions. (Contact MTA for a copy of the Appendix.) These tables were used as the model for the database implementation in the other database packages.

5. Queries

Queries on Oracle were written as PL/SQL procedures. There were two reasons for this: First, it was quite easy to eliminate user input and I/O from timing considerations, leaving only phase 2 as described above. Second, PL/SQL is generally a faster means of retrieving data than entering an SQL statement because PL/SQL programs are stored compiled rather than being interpreted at run time as SQL statements are.[6] See appendix B for the code used to create the procedures. (Contact MTA for a copy of the Appendix.) As with the table definitions, queries were made as similar as possible across database packages.

6. The Database Implementation On FileMan

FileMan is a database package which was originally developed in 1978 by a group of programmers working at the VA under the direction of George Timson.[7] The 21st version of FileMan, including, for the first time, a query optimizer, was recently released. FileMan was originally developed on an older version of MUMPS than currently exists. The structure reflects this. As FileMan is in the public domain, the code can be viewed directly. More modern additions to the language such as "$Get" and "New" appear only occasionally.

Each table is considered to be a "File". Provision is made for repeating fields, something not allowed in the relational model.

In order to compare FileMan performance as closely as possible to its Oracle equivalent, Files were constructed for each of the Oracle tables. In order to make relational joins possible, an additional file was created consisting of nothing but the patient ID, followed by pointer fields to each row in each File corresponding to the same patient ID.

The FileMan database looks something like this:

Pointer Table (PtrTable)
PatientID

Table One	Table Two	Table Three
Where	Where	Where
ID=PatientID	ID=PatientID	ID=PatientID

This structure draws inspiration from an article by Tami K. Winn and Maureen L. Hoye.[8] It proved to be quite easy to simulate the relational "joins" of Oracle using this structure with the added advantage that joins on the PatientID field were implicitly defined. A subtle difference between Oracle and FileMan is that, by default, selected data which includes one or more null fields is not output in Oracle (an inner join). In FileMan such data is an outer join. The global ^DIZ is the default global used for storing data from user files. The first entry for the FileMan equivalent of the Oracle Patient table is therefore:

```
^DIZ(1000,1,0)=100^WAGER,JOHN                    M^111-11-
1111^M^44828^^^7^802-000-0000^625000
```

A single index was created by default: The index for this entry is:

```
^DIZ(1000,"B",100,1)=""
```

which may be interpreted as patient ID 100 points to the first record.

Note the use of numbers instead of strings for subscripts. FileMan started out using exclusively numbers instead of strings because it was originally written before string subscripts were added to the language. The entry in the pointer table pointing to this entry looks like:

```
^DIZ(1001,1,0)=100^1^1^
```

Piece two of the data points to the corresponding entry in the Patient file.

There are three different options in FileMan for the retrieval of data: Print File Entries (option 2), Search File Entries (option 3), and Inquire To File Entries (option 5). Search can be used to do anything the other two can do but appears to be the most resource intensive.[9]

The equivalent of query five is shown here:

VA FileMan 20.0

```
Select OPTION: ?
```

ANSWER WITH OPTION NUMBER, OR NAME

```
CHOOSE FROM:
   1  ENTER OR EDIT FILE ENTRIES
   2  PRINT FILE ENTRIES
   3  SEARCH FILE ENTRIES
   4  MODIFY FILE ATTRIBUTES
   5  INQUIRE TO FILE ENTRIES
   6  UTILITY FUNCTIONS
   7  OTHER OPTIONS
   8  DATA DICTIONARY UTILITIES
   9  TRANSFER ENTRIES
```

Select OPTION: 3 SEARCH FILE ENTRIES

OUTPUT FROM WHAT FILE: PtrTable//

```
.    A- SEARCH FOR PtrTable FIELD: 1:2
.    A- CONDITION: 1 NULL
.    B- SEARCH FOR PtrTable FIELD:
     IF: A//     1:2 NULL
     SORT BY: PtrPatientID//
     START WITH PtrPatientID: FIRST//
```

FIRST PRINT FIELD: 1:1

```
   By '1', do you mean PtrTable 'PatientPtr'?
   YES//    (YES)
   By '#1', do you mean Patient 'PName'?
   YES//    (YES)
```

THEN PRINT FIELD: 2:2

```
   By '2', do you mean PtrTable 'RecordPtr'? YES//
   (YES)
   By '#2', do you mean Record 'ROperatorAct'? YES//
   (YES)
THEN PRINT FIELD:
HEADING: PtrTable SEARCH//
STORE PRINT LOGIC IN TEMPLATE:
PtrTable SEARCH
```

It proved to be much more difficult to isolate phase 2 in FileMan, because of the interactive nature of this program and its heavy reliance on the construction of customized strings which are then executed using the "Xecute" command. It was necessary to alter the actual database code to eliminate I/O and to insert timing statements. The solution was to take a "snapshot" of the local variables and globals used to implement the queries and read/write statements. Routines were written for each of the queries overwriting each of the essential variables with strings that had been edited to remove all read and write statements. The Global ^UTILITY and various local variables were overwritten in the FileMan routines DIP5 and DIO2. Timing statements were added in FileMan routines DIP5 and DIO4. Hopefuly, this represents as accurately as possible all database activity except user interaction and console I/O.

7. The Database Implementation on M/SQL

M/SQL was especially interesting because it allows the programmer the option of embedding SQL statements directly into MUMPS code thus providing the best of both languages. In addition, the package provides a good user interface.[10] It was possible to directly compare the embedded SQL feature directly with Oracle performance by setting up M/SQL tables along the lines of the Oracle tables mentioned above. Small MUMPS programs with embedded SQL statements were written in order to compare query performance. Here is the M/SQL implementation of query seven:

```
query(name)
      ;3-way join based on non-indexed field.
       ;
      new gdata,rdata,pdata
      &SQL(SELECT
GName,ROperatorActivate,Patient    INTO
      :gdata,:rdata,:pdata
      FROM Guarantor,Record,Patient
      WHERE Guarantor=Patient AND Record=Patient
AND PName IN (SELECT PName FROM Patient
      WHERE PName=:name))
      quit
```

8. The Database Implementation on the Ad-Hoc Database

This implementation is an attempt to show what the performance of a customized MUMPS database might be. The ^PT global was adopted as is, and a series of small routines were written to retrieve the same data from it as that of the queries on the other databases. Such customized routines might be written for queries where the performance must be at a premium. The equivalent of Oracle query 8 is shown:

```
query8(id)
        ;
      new GData,RData,PData,PEData,x
      s PData=$g(^PT(id,0))
      if $p(PData,"^",1)=""  quit
      s
GData=$g(^PT(id,4)),RData=$g(^PT(id,3)),PEData=
      $g(^PT(id,6))
      set x=$p(GData,"^",3),x=$p(RData,"^",2)
      set x=$p(PData,"^",1),x=$p(PEData,"^",1)
      quit
```

9. Performance Analysis

The queries developed for testing are briefly described below. Some queries were used for more than one test. See also Appendix A. (Contact MTA for a copy of the Appendix.)

Test:	Purpose:
1 (query 1)	Retrieve the name field from the Patient table (an indexed based search).
2 (query 1)	Same as 1 only this is an unsuccessful search.
3 (query 2)	Get the PatientID from Patient searching on an input name. (non-indexed)
4 (query 2)	Same as 3 only this search is unsuccessful.
5 (query 3)	List the IDs of all Patients whose Social Security Number field has a value of NULL. (full table search)
6 (query 4)	Get Activating Operator (field two) from the Record table and Patient name from Patient.
7 (query 5)	Same as 6 only the Social Security must be NULL.
8 (query 6)	Same as seven with Guarantor Name from Guarantor being added (data retrieval from 3 tables).
9 (query 7)	GName from Guarantor ROperatorAct from Record Patient ID from Patient where the name of Patient matches an input string.
10 (query 7)	Same as 9 only no match is made.
11 (query 8)	Gname from Guarantor ROperatorAct from Record PName from Patient PEEmployer from PEmployer where the id field matches an input number
12 (query 8)	Same as 11 only no match is found
13 (query 9)	GAddressL1 from Guarantor ROperatorAct from Record PName from Patient PEEmployer from PEmployer for all male Patients (PSex='M')
14 (query 10)	GAddressL1 from Guarantor ROperatorAct from Record PName from Patient PEEmployer from PEmployer where the PSocialSecurity field in Patient is not NULL and the FVeteran field from PFather is 'Y'
15 (query 11)	Aggregate function returning the sum of all Patient IDs (well, I had to think of something).
16 (query 12)	Count the number of male Patients whose Fathers are veterans (PSex = 'M' and PFather.FVeteran = 'Y')

10. Results:

A total of 8 runs of these 16 queries were conducted on each database. Each run consisted of four repetitions of the same sequence of queries. The size of each database was increased in increments of 5,000 records. The first database, therefore, was based on 5,000 records; the second on 10,000; the third on 15,000 up to 40,000. The results follow but should be prefaced with remarks on some unique characteristics which were discovered in the performance of each database package. These are summarized below:

M/SQL Results:

M/SQL results are incomplete as of the writing of this article. This is because initial results indicated that there would not be time available to complete all runs for this database package. The results of three runs done with minor changes after each one are shown below. Each run was done five times. Each result shows the average time in seconds needed to complete the query based on the five executions of the run and the query for this run. The line below each result shows the standard deviation, in seconds, of the five executions for each query. In query 8, for example, run one yielded an average execution time of 17010 seconds and a standard deviation of 33 seconds. In run two, this average time went down to 192 seconds and a standard deviation of 0.25 seconds. In the final run the execution time went down to 3 seconds and a standard deviation of 0.1 seconds.

Query Test	Run One	Run Two	Run Three
1	0.536452	0.231981	0.234678
	0.000342	0.001117	0.002642
2	0.244356	0.236292	0.237981
	0.00672	0.004806	0.007647
3	22.286717	1.703751	1.973717
	0.3765	0.018318	0.235389
4	17.387702	1.696480	2.114602
	0.94321	0.022988	0.432771
5	17.046481	1.735170	2.063224
	0.084321	0.013759	0.296699
6	34.254476	3.260471	3.630660
	1.234156	0.015173	0.014569
7	16724.069 26	191.55050 2	2.542191
	11.342156	0.628771	0.144172
8	17010.261 76	192.63787 8	3.022954
	32.784324	0.252939	0.101950
9	1425.9564 32	96.465887	0.363332
	10.43256	0.169739	0.020138
10	1459.9738 50	1510.6754	1.851605
	13.46433	0.12435	0.034866
11	6.222821	1.307689	1.402059
	0.485533	0.008173	0.030974
12	4.921797	1.315837	1.398281
	0.134256	0.007716	0.051220
13	19784.808 74	165.42830 7	3.727501
	34.623341	0.170953	0.101309
14	28863.683 07	256.65976 4	56.896467
	54.788234	0.951355	2.278988
15	19.873174	1.966264	2.020634
	1.342566	0.023539	0.028052
16	3021.2842 34	54.859249	55.579616
	43.522667	0.706744	0.079765

Explanation:

The first run shows the results of nearly precise equivalents of the Oracle PL/SQL queries being run on M/SQL.

The second run was done after consulting with Intersystems, the product owner. I realized that I had neglected to provide the M/SQL query optimizer with accurate statistics on the nature of the database. The database administrator should provide the optimizer with approximate figures on the number of rows in each table and the number of unique values in each field. This information was updated in the data dictionary and the same query sequence was run again.

The last run was done after a careful look at the SQL code written to do the queries. As can be seen, running time has gone down significantly. Query tests nine and ten especially show dramatic improvement. Tests nine and ten were run using the code in routine ^query7 which can be seen above. The problem with the original code was that the running time could potentially be O(N^2) as first the fields are selected from the Guarantor, Record and Patient tables based on Patient ID, which is an indexed field, and then the entire Patient table is searched looking for a match for the parameter name. This could be followed by potentially N repetitions of this sequence of actions for a name with no match in the database. Rewriting this code reduced this to an O(N) operation:

```
SELECT Gname, ROperatorAct, Patient
FROM Guarantor, Record, Patient
WHERE Guarantor=Patient AND Record=Patient
AND Pname=:name;
```

In general, the M/SQL optimizer tends to perform better on select statements which do not use embedded select statements. For example:

```
SELECT name, id from Patient
WHERE id IN
(SELECT id FROM Admitted);
```

This statement could be rewritten as:

```
SELECT name, id FROM Patient, Admitted
WHERE Patient.id = Admitted.id;
```

The staff at Intersystems indicated that it is possible to provide "hints" to the M/SQL optimizer by using the ORDER BY command to tell the optimizer which fields to search by first. ORDER BY is used to indicate which fields to sort by. If the above statement was sorted by name, for example, I would write:

```
SELECT name, id FROM Patient, Admitted
WHERE Patient.id = Admitted.id
ORDER BY name;
```

Oracle Results:

The Oracle cache may play quite a substantial role in query performance. There are two observations which lead me to conclude this. The first is that performance varies quite considerably from the first repetition of each query sequence to the second. An extreme example is a run where query test one took 15 seconds on the first repetition and 1.5 seconds on the second. Similar results may be seen when a program is first loaded into memory on any cache based machine. In this case it is the actual queries which are not present in the cache at the start of each run. The second observation is that the standard deviation for the queries is quite high for the oracle run statistics. The standard deviation was as much as 50% of the average for some. It was thought at first that the results had been skewed by the minimal operating system activities allowed. The standard deviation for the other database packages is, however, nowhere near the level of the Oracle statistics, and it must be concluded there is some factor at work here not present in the other database packages.

FileMan Results:

FileMan shows increased time needed in terms of the number of fields involved (not tables). This may be due to the fact that the modified version of FileMan goes through all the motions of printing the data retrieved to the console without actually doing the system I/O, and the costs of printing these fields may actually outweigh the cost of searching for them. For queries one and two the FileMan option "INQUIRE TO FILES" could have been used as well as the standard query option "SEARCH FILE ENTRIES". Option 2 "PRINT FILE ENTRIES" was used for tests 6, 11 and 12.

Ad-Hoc Results:

Of the four databases, these results are the most uniform and predictable which is not surprising since there is very little in the way of code beyond the bare minimum needed to execute the query here.

Results:

Please see the appendix following this paper. (Contact MTA for a copy of the Appendix.)

Conclusion:

Which database package performed the best? Before drawing any conclusions it should be noted that each database package *could* have performed much better. The astute observer will notice that each database is allowed only one indexed field, namely the unique identifier of each table row. Whenever

possible the default options were chosen when each database was set up. For the ad-hoc database, performance would have been better if more of the data had actually been in the subscripts.[11] This was done for the reasons mentioned in the introduction. The initial approach to each package hopefully reflects what any first time user might do: address the big picture first and fill in the details later. I was also interested in just how easy it might be to gain acceptable performance. The fastest database package in the world will be of limited benefit if it requires an enormous amount of knowledge to fine tune it. With the knowledge gained from this first round of performance analysis, the goal will be trying to make each database run as fast as possible in the future.

Clearly the ad-hoc database performed the best overall. This indicates that a customized database written by a programming team is still the best solution if performance is sought at all costs. Unfortunately, the issue is not that simple. Databases change in nature over the years, and what may have been a good solution five years ago may no longer work today. It would be easier to buy the latest version of a database package such as FileMan and to change the database with the help of this package rather than maintain the original programming team. Nevertheless, this sort of customized programming may be necessary if some queries must absolutely perform within a specified time.

M/SQL results indicate that better results may be achieved where careful attention is paid to the text of the code and to the maintenance of statistics for the optimizer. M/SQL results can vary quite widely based on the degree to which this is done. An automated tool to update the statistics on the current state of the database would help to improve performance and would take some of the burden of maintaining optimizer performance off the database administrator. The following code was used to find the number of rows and unique field values for the Patient table. Something like this could be generalized into a utility routine which could then be used by the database administrator to provide the query optimizer with *exact* statistics on the state of the database:

```
Start(PC)
;PC is number of pieces in the data section.
;
new hl,j,RC,Uniq
;
;initialize the unique count for each field.
for hl=1:1:PC s Uniq(hl)=0
;
set hl=$o(^Patient(2,0,"")),RC=0
;get first element
f  s hl=$o(^Patient(2,0,hl)) q:hl=""  s RC=RC+1  d Dist
;
;print results:
write !,"There are ",RC," rows in ^Patient"
f j=1:1:PC w !,"There are ",Uniq(j)," values for field
",j
kill ^Temp
;
;main routine ends here
            quit
Dist
      f j=1:1:PC s h2=$p(^Patient(2,0,hl),$c(1),j)  i
      $d(h2)  d Ch
      quit
Ch
      if h2=""  quit
      i $d(^Temp(j,h2))=0 s^Temp(j,h2)="",Uniq(j)
      =Uniq(j)+1
      quit

/**********Run Results **************
2d5>do Start^GetStats(10)
There are 1000 rows in ^Patient
There are 993 values for field 1
There are 632 values for field 2
There are 3 values for field 3
There are 967 values for field 4
There are 5 values for field 5
There are 6 values for field 6
There are 21 values for field 7
There are 837 values for field 8
There are 995 values for field 9
There are 0 values for field 10
12d5>
```

If possible, an effort should be made to make run time less dependent on the input code. My experience with query seven is something which could happen to the user writing an "on the fly" query.

It is more difficult to say which of the two packages, FileMan20 or Oracle, performed better. Oracle performance was outstanding on test 3 and 4, a search retrieving a single field from a single table based on a non-indexed search. A chart of the performance of the respective databases here follows:

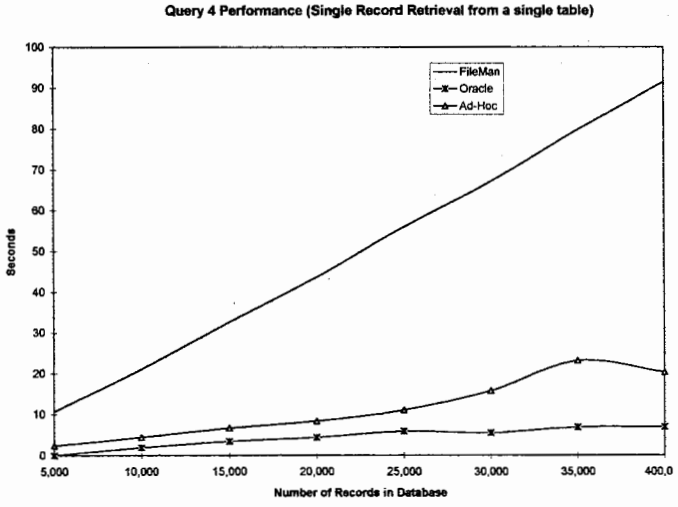


Chart 1

Possibly Oracle builds an index on this field in memory which it then uses in subsequent queries to reduce running time.

Oracle performance declines markedly as the number of records retrieved goes up:

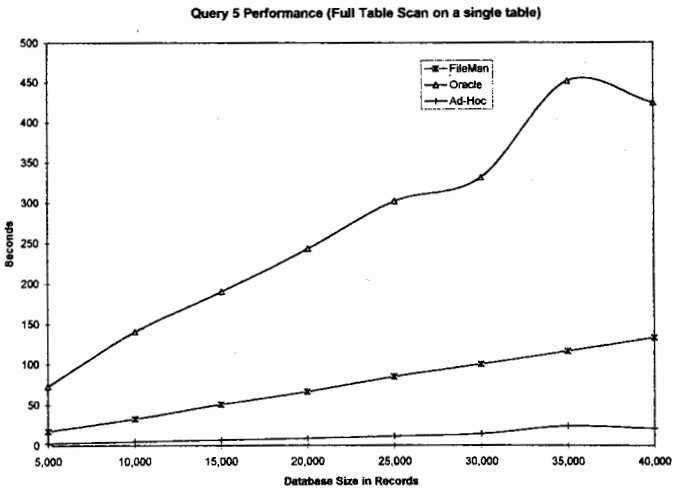


Chart 2

The final chart shows another characteristic of Oracle: As the complexity of the query goes up performance also goes down:

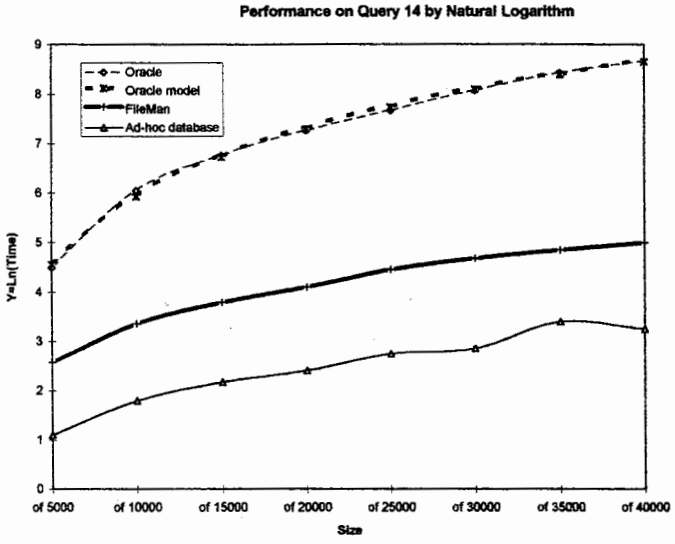


Chart 3

The model curve is based on the function f(x)=0.00000049158 * X to the second power, which was determined using the Least Squares Method.

This indicates a possible optimization strategy for Oracle which will be pursued in the next part of this study: If Oracle does well on simple queries and poorly on complex ones, then perhaps the larger queries could be broken down into smaller ones.

It should be mentioned here that certain queries are bound to be performed more often than others. The average performance of a database should be a weighted average of the queries presented with those performed frequently receiving the most weight. Many of the queries I have designed would not be performed more often than once a week. Certainly, a query such as query one, requesting information on a specific patient, would be performed quite often. Hierarchical performance here is simply not to be outdone. Thus for the every day operations on a database, FileMan may perform *better* than Oracle. In addition, the programmer is in much more control with the FileMan or M/SQL database. If it is necessary to write a customized routine which goes directly to the data structure this can be done. With Oracle, it is always necessary to work through the data management system.

Some observations on FileMan:

It is difficult to modify and/or debug FileMan code. This is due to of the extensive use of "Xecute" and of naked globals. Often strings to be executed contain an embedded xecute command. It was often difficult to determine which variable was being referred to. The extensive use of "xecute" may also have some effect on performance. It is unfortunate that string subscripts are not made more use of. This would also cut down on execution time.

Some observations on Oracle PL/SQL:

The following procedure will compile without errors. If this procedure is run it will crash the system:

```
CREATE  OR REPLACE PROCEDURE query_bug(PPatientID
      IN NUMBER) IS
      name_data CHAR(30);
BEGIN
      SELECT PName INTO name_data FROM Patient
      WHERE PPatientID = PPatientID;
END query_bug;
```

The reason this query fails at run time is that the parameter passed in matches a field name in the table being searched resulting in an ambiguity the system cannot handle. Apparently a PL/SQL procedure with this problem actually disrupted an organization's processing for many weeks.[12]

The following query will compile *and* run but will not produce the desired results:

```
SELECT PName FROM Patient WHERE PSocialSecurity =
 NULL;
```

The proper syntax for the information desired is "IS NULL" not "= NULL".[13]

These are some of the hazards confronting the user of PL/SQL today. In addition, the programming environment is not

a friendly one. There is no debugger and it took a day or two to figure out how to do simple screen output for debugging purposes because screen output is not an integral part of PL/SQL.[14]

All the packages studied thus far have had their advantages and drawbacks. What came as a surprise, was how easy it is to make a hierarchical database package perform as a relational one. The relational model is a quite powerful one and has had a great deal of influence on database design in recent years. Perhaps the flexibility of this model can be combined with some of the performance advantages the hierarchical storage system offers in order to capture the best of both worlds. My special thanks to the InterSystems Corporation, the IDX Corporation and the Department of Veterans Affairs for all the help and advice given in completing this study.

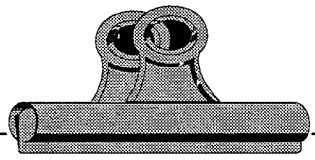
Warren G. Weis is completing the degree requirements for a Master degree in computer science at the University of California, Davis.

End Notes

[1]C. J. Date *An Introduction to Database Systems*. page 269.
[2]Database Protocol from IDX. Faxed on September 21st.
[3]*Database Systems* Vol 1, Part III by C.J. Date contains a good description by the originator of the relational model. Other sources I have found to be quite good are *Database System Concepts* by Henry F. Korth and Abraham Silberschatz and *Fundamentals of Database Systems* by Elmasri and Navathe.
[4]Dr. Richard Walters. Conversation.
[5]Oracle RDBMS Database Administrator's Guide, Vol 1, Chapter 1
[6]*Mastering Oracle7 & Client/Server Computing* by Steven M. Bobrowski. page 123.
See also Oracle Manual on PL/SQL chapter one.
[7]Transcript of an interview with George Timson conducted members of the San Francisco ISC staff on May 7th, 1991.
[8]*Relational Features of VA FileMan*. Tami K. Winn and Maureen L. Hoye.
[9]See Page 59 of the *VA FileMan 20 User Manual*
[10]Mouse support is expected to enhance the interface in the near future, however. Bob Chapski @Intersystems. Conversation.
[11]See Walters, Richard "Database Optimization: An Overview" in MUMPS COMPUTING, vol 22, no. 5.
[12]*Oracle Performance Tuning* by Peter Corrigan and Mark Gurry. pg.137.
[13]Oracle's treatment of NULL conforms to ANSI standards which state that:
NULL is never equal (=) to anything.
NULL is never NOT equal (<>) to anything.
Any relational comparison with NULL such as less than or (<) or greater than (>) is always false.
Oracle Performance Tuning Page 150.
Perhaps it would be helpful, however, for the compiler to issue a warning that no rows will be returned. This might avoid some potential misunderstandings.
[14]This involves turning on the terminal output environment variable ("set serveroutput on") and using the "putline" procedure from the package dbms_output. To write the string "Hello world" onto the terminal screen one would , for example, write the code line "dbms_output.putline('Hello world')."

List of References

Bobrowski, Steven M. *Mastering Oracle7 & Client/Server Computing*. Alameda, CA: SYBEX, Inc., 1994.
Corrigan, Peter and Mark Gurry. *Oracle Performance Tuning*. Sebastopol, CA: O'Reilly & Associates Inc, 1993.
Date, C. J. *An Introduction to Database Systems*. vol. I 5th ed. Reading, MA: Addison-Wesley, 1990.
Elmasri, Ramez and Shamkant B. Navathe. *Fundamentals of Database Systems*. 2nd ed. New York: Benjamin/Cummings, 1994.
Korth, Henry F. *Database System Concepts*. New York: McGraw Hill, 1986.
Lewkowicz, John. *The Complete MUMPS*. New York: Prentice Hall, 1989.
ORACLE Developer's Release Documentation. Version 7.0. Redwood City, CA: May 1992.
VA FileMan Developer's Release Documentation. Version 20.0. San Francisco: Information Systems Center, July 1993.
Walters, Richard. "Database Optimization: An Overview." *MUMPS Computing* 22 no.4: 52-59.
Winn, Tami K. and Maureen L. Hoye. *Relational Features of VA FileMan*.



Call for Articles

Here at MTA we are always looking for information that is interesting and useful to our members. If you have developed a useful M application; found a better or more cost effective way to do something in M; reduced overhead or boosted productivity; or simply have a useful M tip, we'd like to hear from you.

Simply submit an abstract describing your idea to:

Managing Editor
M Computing
1738 Elton Rd.
Suite 205
Silver Spring, MD 20903-1725

All abstracts and subsequent papers will be reviewed and judged appropriate for publication based on content, originality, accuracy, and usefulness to the M community.

If you are chosen for publication, you will be notified by MTA.

Don't miss your chance. Contribute to *M Computing*. Submit your ideas today! -- the Editor