MAXIMIZING PERFORMANCE FOR IMAGE DATA BASES

Kathy S. McKittrick
Dynamic Information Systems Corporation
910 15th Street - Suite 640
Denver, Colorado 80202

What exactly is the value of the data that is contained in the data bases on your system? How can you measure that value and what components contribute to the equation?

For years we data processing professionals have been busy developing systems to capture and disseminate information. We do this based on user requirements and requests, often times without understanding or examining how those systems will effect our company's success.

Why is it important that we understand this? Put more specifically, why is it important that a programmer understand the value of a report that he or she is writing a program for? There are two reasons. First, the programmer stands a much better chance of fulfilling the user's requirement if he/she understands its purpose. The communication between a user and a programmer is often challenging and an understanding of the business problem being solved enhances that communication.

Secondly, the programmer's job satisfaction will be greatly enhanced by understanding the use of his/her finished product. If a programmer is simply juggling bits and bytes around, the job becomes like that of an assembly line worker who bolts on door handles, but never gets to see the finished product. We all enjoy our jobs better when we can see the tangible fruits of our labors.

So the first step is understanding the business problem(s) being solved. The next step is quantifying the value of solving that problem.

Let's agree on one basic, simple truth: that users benefit from the information they GET OUT of the system, not the information that they put into the system. "Wait a minute!" you might say, "You can't get any information out of the system until its been entered into the system!" True enough! But entering information into the system is not the goal. It's the necessary evil. You can enter customer information, orders, invoices, and receivings all day long but unless you have a good mechanism for retrieving that information in meaningful formats, all that entry is useless. Most computer systems contain vast amounts of data; much of which is difficult or impossible for users to get at. And how ironic! If users can't get to it, it has no value!

Let's talk about some of the types of information that are "locked" inside computer systems and data bases. Then we'll talk about why its so difficult to get at and some of the strategies that we can employ to do a better job of making information available.

INFORMATION THAT IS "LOCKED" INSIDE YOUR SYSTEM

During this discussion, be thinking about how our hypothetical company (ACME Distributing Company) may have some similarities to your own company, at least in-so-far as applications go. ACME Distributing uses an HP3000 series 68 for processing orders, processing shipments, invoicing, accounts receivable and accounts payable. There are fifty order entry operators on the system during business hours and if things are going well they are entering orders throughout the day. There are four terminals in shipping so that orders can be updated with a "shipped" status as soon as they've left the loading dock. There are five receivables clerks and five payables clerks using the system fairly consistantly during the day to enter payments, request checks, and do collections.

There are also ten executive and mid-level managers who have on-line access to the system. A couple of years ago the MIS department purchased a report writer so these management users could easily (meaning without a programmer), extract information from the system. Prior to using the report writer, management had received reports from the system, however these often included too much detail or a summary of all information. Typically, management wants to see only the exceptional information.

About six months ago, the MIS Director had a meeting with the executive staff to discuss system performance. After many complaints from users on the system being too slow, an investigation was done to discover the cause of the bottle neck. The MIS Director reported that two things had to be done to improve performance:

1. Reports run by management often required serial reads and these reports would now be run only at night.

2. The series 68 machine must be upgraded to a series 70.

It turns out that the management reports were already taking from 20 minutes to two hours to run and managers impatient for faster results, were disappointed to hear they were the ones being knocked off of the system. But they also understood the wisdom of servicing customers as quickly and efficiently as possible, so they didn't complain too much.

They now had to wait a full day to get answers to their questions; questions that sometimes affected what inventory was purchased or moved from one location to another. They soon found that this also affected customer service. The management information so critical to doing business on an hour-by-hour basis was "locked" inside their computer system and unavailable to those who would benefit from it.

This is a common story in our industry. High priced decision makers are prohibited from getting timely information upon which to base their decisions. The priority is given to ENTERING the information when the prime benefit of entering the information is BEING ABLE TO RETRIEVE IT.

Anytime you say to a user, "I'm sorry, you can't run that during the day", you may be impacting your company's bottom line or their ability to do business.

The Sales Manager at ACME, for example, heard one day from a sales representative that a large order for blue pens would be placed later in the week by an important customer. The Sales Rep stressed that it was important that ACME be able to deliver the pens the next day. There wasn't enough stock in the local warehouse so the Sales Rep wanted to know if the Sales Manager could have inventory shipped in from other locations. This required the Sales Manager to run a report which looked at blue pen inventory in all twenty area locations, subtract inventory that had been committed during sales that day, and predict how much could be shipped without impacting the other locations. This report was DISALLOWED during the day.

So, the Sales Manager set it up to run that night. Unfortunately, he entered the part number incorrectly and when he came in the next morning he discovered that nothing came out on the report. Even though it had run for two hours.

Back to the drawing board. He resubmitted the job to run that night. Now he'd have his answer on Wednesday. (He was more careful about entering the part number this time).

Wednesday morning. The important customer's order comes through. The Sales Rep is on the phone to the Sales Manager. The report has come through but just a little too late. It turns out that inventory has been committed at the alternate locations and can't be shipped to the important customer. The Sales Manager has already called the factory and they can drop-ship the pens to the customer by Friday afternoon. Not good enough. The Sales Manager has a decision to make. Either divert the inventory in stock to the important customer and make many smaller customers angry or deliver to the smaller customers as promised and risk losing the important account.

And all because ACME was trying to save on system performance. If the Sales Manager could have run the report on Monday morning and again on Monday afternoon after his initial error, the manufacturer could have delivered in time.

When situations like this occur it becomes clear that we must find a way to provide timely information and there is always a solution. It becomes a question of "cost versus benefit" not "possible versus impossible".

WHY IS IMPORTANT INFORMATION LOCKED IN???

We've talked about system performance as a culprit in making critical information unavailable. But what are the underlying reasons for the poor performance?

There are many. We'll talk about three key areas in this paper but understand that in complex multi-user, multi-application systems, there are no simple answers. Each individual component of a system must be examined as-well-as taken as a part of a whole.

The three areas we'll focus on here are:

- Data Base Design
- Volume of Data
- Available Indexing/Retrieval Methods

Data Base Design

The design of a data base that is being used to provide information to users on a regular basis can have an enormous impact on performance and therefore, the users' ability to retrieve information. For example, if we need to obtain information from fifteen different data sets for a particular report, we're going to be expending far more disc I/Os than if we retrieved information from only one data set. And in a reporting or inquiry application disc I/O is generally the most significant factor in the performance equation.

The single biggest reason for poor data base design is that we try to use a single data base for too many purposes. We cannot for example, optimize a design for transaction

DESIGN FOR THROUGHPUT

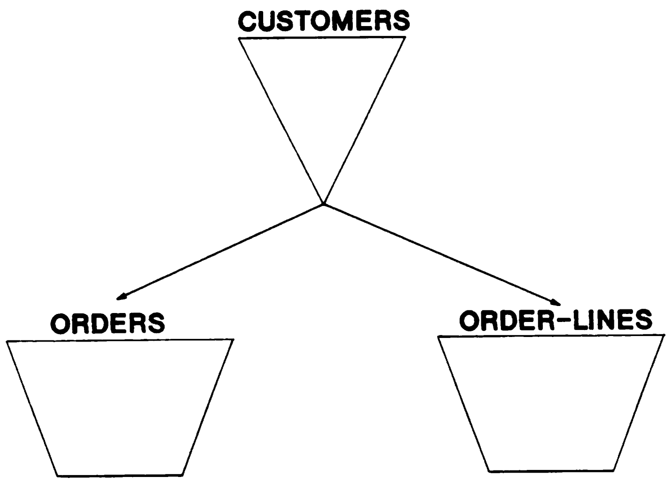


FIGURE 1

throughput and data retrieval both. By trying to use one design to solve both problems we wind up with a design that is not optimized for either task.

In a sales application for example, if we were designing to optimize throughput with no thought to retrieval requirements, we might have a design similar to the one you see in Figure 1. We have a customer master containing name and address information, an order detail containing information such as the date of the order, the total dollar amount of the order, bill-to and ship-to information, and an order line detail, with one record for each line on the order. This data set would contain the inventory item number, unit price and quantity information. Notice in this design I have included only one path to each detail data set; the ACCOUNT-NUMBER of the CUSTOMERS master links to each detail set. Remember, I've designed this data base in order to optimize throughput (adding records), and as a result I've minimized the I/O overhead when adding records by limiting the number of paths.

It is important to note that each path on an IMAGE detail data set requires an average of four I/Os each time a record is added or deleted.(1) By minimizing the number of paths on a set, you reduce update overhead.

So adding records to the ORDERS and ORDER-LINES data sets will be extremely efficient.

Now let's look at a second data base design. This one will be optimized for retrieval purposes and will include only the ORDERS data set. The ORDERS data set contains the following fields:

ACCOUNT#	(the customer's account number)
DATE-ORDERED	(the date the order was placed)
SALES-REP	(the Sales Representative's initials)
REGION	(the sales region)
STATUS	(the status of the order, ie., shipped, back ordered, invoiced, etc.)
ORDER#	(a sequentially assigned sales order number)
ORDER-TOTAL	(total dollar amount of the order)
STATUS-YRMO	(a concatenation of the status field and the year and month of the order)

We'll further assume that our users need to select records based on DATE-ORDERED, SALES-REP, REGION, STATUS, and STATUS-YRMO. Given this scenerio, this "Informational" data base would probably be designed as shown in Figure 2. Remember, we're not concerned here with update overhead, just retrieval capabilities. So we've provided our users with a path for each field that will be used as selection criteria.

(1) Robert M. Green, F. Alfredo Rego, Fred White, David J. Greer, Dennis L. Heidner, *The IMAGE/3000 Handbook*, (Wordward 1984).

DESIGN FOR RETRIEVAL

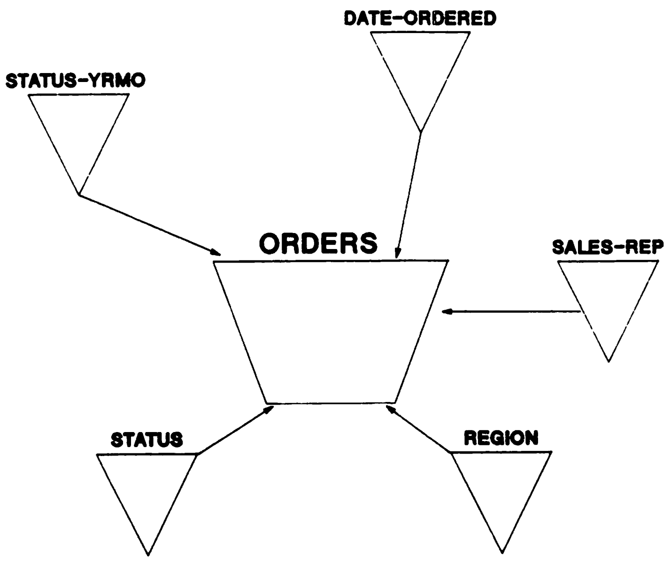


FIGURE 2

THE COMPROMISE

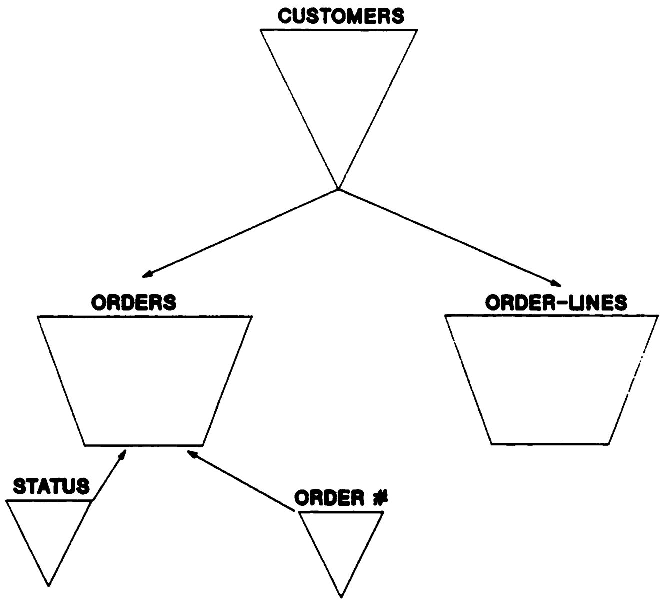


FIGURE 3

It is easy to see that these two data base designs are very different due to the fact that they are being used for two different purposes. But this is not the way that we usually design systems. Typically we come up with a data base design that is a compromise between our retrieval and update requirements. In our order entry data base example, it would look something like the design in Figure 3. We've provided our users with the bare minimum in terms of paths in order to minimize the update overhead. This means that due to many retrievals the ORDERS data set will have to be serially read. We may have to limit our users access to the information.

In addition, as we add orders, we're going to be incurring more overhead due to the additional paths that we decided are essential. The compromise data base is less efficient for updating AND retrieving information.

To compare the efficiency of a compromise data base versus a "split" data base approach we'll run through a possible scenerio. Refer to Figures 1, 2, and 3 for design information. To do this we'll make some basic assumptions:

1. 600 orders are entered into the system on a daily basis.
2. An average order includes one ORDERS entry and ten ORDER-LINE entries.
3. There are 250,000 entries in the ORDERS data set.
4. The ORDERS entries are 256 bytes in length and are blocked at five records per block.
5. There are 20,000 shipped orders (STATUS of "SH").
6. There are 2,000 orders that were entered in March of 1987.
7. There are 200 orders that were entered in March that are in a shipped status.
8. A report is run daily selecting orders that have shipped in the current month.

First, we'll run through the numbers for the compromise data base. In order to calculate the I/O required to enter the orders, we'll use the following equation:

```
        600 x (3 + (4 x 3)) =  9,000 I/Os  (ORDERS data set)

(10 x 600) x (3 + (4 x 1)) = 42,000 I/Os  (ORDER-LINE data set)

                      TOTAL  51,000 I/Os to enter orders
```

Remember that our formula for I/Os to add detail data set records is 3 + (4 x #paths). One ORDERS record and ten ORDER-LINE records are used per order.

Now, to provide the report (on all orders shipped in March), we need to use the STATUS path of the ORDERS data set. Since STATUS is not likely to be the primary path for this set, we assume that to read each record in this chain requires one I/O, and performing the DBFIND to find the chain head for the "SH" (shipped) path costs one I/O.

```
DBFIND for chain head          1
DBGETs to read chain      20,000
                TOTAL     20,001
```

In summary, to enter orders on our "compromise" data base costs us 51,000 I/Os per day and to run the report costs us 20,001 I/Os.

```
Entry       51,000 I/Os

Retrieval   20,001 I/Os
  TOTAL     71,001 I/Os for daily processing
```

Now let's look at what it takes to accomplish these same tasks using our split data base approach. In this case, we have only one path on the ORDERS data set, so the calculation for adding orders would be:

```
      600 x (3 + (4 x 1)) =  4,200  I/Os (ORDERS data set)

(10 x 600) x (3 + (4 x 1)) = 42,000  I/Os  (ORDER-LINE data set)

                   TOTAL  46,200  I/Os to enter orders
```

To update the data base designed for retrievals (we'll talk about how to do this later) the calculation would be:

$$600 \times (3+(4 \times 5)) = 13,800$$

To run the report from our data base designed for retrievals, we can use the STATUS-YRMO path, and read only the 200 records that qualify for our selection criteria. Adding the one I/O for the DBFIND, this gives us:

```
DBFIND for chain head          1
DBGETs to read chain         200
                TOTAL        201
```

The total number of I/Os for daily processing on the split data base approach is:

```
   Entry    46,200 I/Os
   Update   13,800 I/Os
Retrieval      201 I/Os

    TOTAL   60,201 I/Os for daily processing
```

And how does that compare with the I/Os it took to process on the compromise data base?

```
Compromise data base     71,001 I/Os
Split data bases         60,201 I/Os

                         10,800 I/Os  DIFFERENCE!!!!!
```

By splitting the data bases we reduce our I/O overhead by 15%! We will increase the amount of disc space used, but disc space is a one time cost whereas the cost of slower processing occurs over and over again. Just imagine improving your own system performance by 15%!

Let's look at what the disc space differences might be for the sample data bases. First the compromise data base. For simplicity sake, let's assume that each of our three main data sets (CUSTOMERS, ORDERS and ORDER-LINES) contain media entries that are 256 bytes (or one sector) in length. Let's also assume that we have 10,000 customers, 250,000 orders, and 2,500,000 order lines.

For our auto masters (ORDER# and STATUS) we need to calculate based on the size of each field plus the chain pointers times the number of different values likely to be contained in each set. Let's assume that there are ten possible status codes and that STATUS is a two-byte field. In the case of ORDER#, let's assume an eight-byte field. Since there will be a unique ORDER# for each ORDERS record there will be 250,000 records in this auto master.

```
STATUS      (2 + 12) * 10          =          140 bytes
ORDER#      (8 + 12) * 250,000     =    5,000,000 bytes
CUSTOMERS        256 * 10,000      =    2,560,000 bytes
ORDERS           256 * 250,000     =   64,000,000 bytes
ORDER-LINES      256 * 2,500,000   =  640,000,000 bytes
                                   ----------------------
                                      711,560,140 bytes
                                            or
                                        2,779,532 sectors
```

Now let's compare that with our split data base approach. For the data base designed for throughput, the disc space required would be:

```
CUSTOMERS       256 * 10,000       =    2,560,000 bytes
ORDERS          256 * 250,000      =   64,000,000 bytes
ORDER-LINES     256 * 2,500,000    =  640,000,000 bytes
                                   ----------------------
                                      706,560,000 bytes
                                            or
                                        2,760,000 sectors
```

The disc space required for the data base designed for retrieval would be:

```
ORDERS          256 * 250,000      =   64,000,000 bytes
STATUS         (2 + 12) * 10       =          140 bytes
DATE-ORDERED   (6 + 12) * 220      =        3,960 bytes
SALES-REP      (4 + 12) * 100      =        1,600 bytes
REGION         (4 + 12) * 10       =          160 bytes
STATUS-YRMO    (6 + 12) * 120      =        2,160 bytes
                                   ----------------------
                                       64,008,020 bytes
                                            or
                                          250,032 sectors
```

(NOTE: We are making the following assumptions for the number of records in each of the automatic masters:

STATUS - we established earlier we would assume 10 status codes

DATE-ORDERED- assuming here one years worth of data; approximately 220 work days per year.

SALES-REP - we assume 100 Sales Reps

REGION - we assume 10 regions

STATUS-YRMO - given 10 possible status codes and 12 months, there are 120 possibilities here.

Furthermore, we add 12 bytes to the length of each record to account for the chain pointers and chain count in the master sets).

For both the throughput and retrieval data bases:

```
          2,760,000 sectors
        +   250,032 sectors
        ---------------------
TOTAL     3,010,032 sectors
             versus
          2,779,532 sectors (for the compromise data base)
```

In conclusion, by using more disc space and designing separate data bases for separate purposes, we are providing the business with more benefit from the computer system. The cost benefit when measured in terms of resources looks like this:

COST	BENEFIT
230,500 sectors disc space	15% throughput improvement

There's one issue that we have yet to discuss regarding this split data base strategy. That is keeping the information in the second data base up to date. There are several approaches that you can take.

There are several products on the market which are designed to keep two or more data bases in sync. (Silhouette and BackChat are two that I know of.) In some cases these products allow you to mirror only selected data sets, and at configurable intervals.

If you cannot use a separate machine for informational data bases consider running a nightly process to move records from the "operational" or throughput data base. This approach will save your on-line users from the overhead of updating the informational data base.

Finally, make sure that your operational and informational data bases reside on separate disc drives. If at all possible, providing a separate machine for informational systems is the most effective solution of all. This way there is no impact on the operational users even when updating the informational data base during on-line hours.

<u>Volume of Data</u>

Sometimes the shear volume of data that we must extract information from is a limiting factor, even when the operational and informational data is separate. In the case of the sales order information, it is not unlikely or unreasonable in some businesses to keep three year's of sales information available for management reports.

So, if we keep three year's worth of information in our informational data base, and we need to report on total sales dollars for a particular region in a particular time frame, what are we looking at in terms of reporting time? Let's take a closer look.

Remember that we had 250,000 ORDERS in our data base that contained one year's worth of information. Based on that figure let's assume that three years worth of information would be three times that figure or 750,000 records.

Now we need to select on both the REGION (one of the paths defined in our informational data base illustrated in Figure 2) and the DATE-ORDERED. Let's say, hypothetically, that we're looking for orders in Region 7 during March of 1987.

Based on the fact that there are ten sales regions in ACME Distributing Company, let's assume that 10% of the records (or 75,000) would be orders for Region 7.

Let's further assume that since we have three years worth of history, that approximately 21,000 sales occurred each month. If one-tenth of those total sales occurred in Region 7, then 2100 records would ultimately qualify for our report selection. But how many records do we have to read in order to get to those 2100 records of interest?

Refer again to Figure 2, our data base designed for retrieval. You can see that the only IMAGE path available to us is the "REGION" path. We can't use the DATE-ORDERED path because we're selecting on a RANGE of dates, not specific dates. IMAGE will only allow us to use full key values when doing a chained read.

We can therefore assume that we're probably going to do 75,000 I/Os in order to select the 2100 records of interest. Why will we get no benefit from the blocking factor? Because unless the records are sorted in sequence by region before they are added to this data base, (a scenerio that is highly unlikely), it is unlikely that any of the records for Region 7 will wind up in the same block. Refer to Figure 4. You can see how records for region 7 are spread throughout the detail data set. Since one I/O is performed per block (not taking disc caching into account) and we only get one record of interest per block, we will do 75,000 I/Os to read 75,000 records.

Figure 5 shows the advantage we would realize if we were to make REGION our primary path and reload this detail set in primary path sequence. Because our blocking factor is 5 we would do one-fifth the I/Os after a reload. Products that can perform highspeed single set reloads are Adager, Dbmgr and Dbgeneral.

But is it practical to make REGION our primary path? Only if it is the path that we will most often use to retrieve information. That is the whole purpose of specifying a primary path; to allow for more efficient retrievals. Unfortunately, REGION is unlikely to be a candidate for primary path.

So, in fact, it looks like we'll do approximately 75,000 I/Os to obtain the information we need.

Unless we use some alternative method for retrieving the data. We have two choices:

- Large blocked reads (called MR NOBUF)
- Alternative indexing methods

Why have I not included MPE caching as an alternative? Because MPE caching uses the RANDOM FETCH QUANTUM when doing serial reads of IMAGE data sets, not the sequential fetch quantum. The whys and wherefores of this are beyond the scope of this paper but you can test this yourself as I did. Take a data set within an IMAGE data base and perform a serial read in a batch job (a QUERY "FIND ALL" command will do). Then write all the records in the set out to a flat file. Using the same blocking factor as in the IMAGE data set. (Be sure and set the MPE record size large enough to account for the MEDIA record length, so that you're comparing apples to apples.) Then do a serial read on the MPE file in a batch job. Compare the time it takes for each and you'll find that the read of the MPE file is much faster. This, of course, assumes that your sequential fetch quantum is set fairly high and your random fetch quantum fairly low (as recommended by HP).

So...back to our two alternatives; large blocked reads and alternative indexing methods. The method that you choose here will depend largely on how many records are being selected from the whole and whether or not you need to select from more than one data set.

The way large blocked reads work (these are also known as MR NOBUF or multi-record no buffering) is to read many thousands of bytes of information per physical disc I/O. This method can significantly reduce the time an I/O intensive job takes (such as our report) by reducing the number of disc reads required. In the case of our report, to read the REGION chain took 75,000 I/O's. Now let's look at what a serial read using an MR NOBUF tool such as SUPRTOOL by Robelle would take.

The default buffer size for SUPRTOOL is 14,366 words, or 28,732 bytes. Given that our ORDERS records are 128 words (256 bytes) in length, we can read 112 records per disc I/O. Given that we have 750,000 records in the ORDERS data set, this would mean that we can accomplish our task using 6,697 disc I/Os -- a savings of more than 68,000 I/Os over our chained read! (See Figure 6).

$$\frac{\text{21,000 record pointers for "8703"}}{\text{64 record pointers per I/O}} = \text{329 I/Os}$$

$$\frac{\text{75,000 record pointers for "07"}}{\text{64 record pointers per I/O}} = \text{1,172 I/Os}$$

1,501 I/Os

(to retrieve the records using the relative record numbers) 2,100 I/Os

3,601 I/Os

FIGURE 8

The second possibility we'll look at is alternative indexing techniques. A method available for multi-field selection on IMAGE data bases is OMNIDEX by DISC. Using this method to index the REGION field and the first four characters of the DATE-ORDERED field (which contain year and month) we can select the same 2,100 records using only 3,601 I/Os! Let me explain.

The way this indexing technique works, in simple terms, is to maintain an index of values and their associated record pointers. With IMAGE detail data sets the relative record number is used as the internal record pointer. (In the case of master data sets, the IMAGE search item value is used as the internal record pointer).

Selection criteria (or indexed values) can be words within fields or specific portions of fields, such as year and month from the date field.

When selecting records by two or more selection criteria, OMNIDEX simply reads each set of record pointers into stack and finds the intersection of those two sets. Figure 7 illustrates what a subset of our sales history report selection might look like. This Figure illustrates what the selection of the 2,100 entries for our report entails. There are several record pointers which appear in both lists. These constitute the intersection.

Now it turns out that each physical index record contains up to 64 record pointers. What this means is that 64 record pointers are read per disc I/O, if there is no benefit realized from blocking. (These index sets can be reloaded so that up to seven index records or 448 record pointers are retrieved per I/O). Figure 8 shows the calculations used to arrive at our figure of 3,601 disc I/Os using Omnidex indexing methods to retrieve records for this report.

From this example, you can begin to see how the method used to improve performance is dependant upon the number of records that are ultimately used. Using sophisticated indexing techniques was by far the best way to perform this retrieval. In fact, half the number of I/Os were used as compared to the number used for an MR NOBUF serial read.

But let's look at another example that shows a different result. If instead of looking for one month's sales for Region 7, we were looking for one year's sales, the numbers would look different for the indexed retrieval.

We stated earlier that the 750,000 ORDERS records represented three year's worth of sales. Let's assume, then, that in 1987 there were a total of 250,000 orders, or one third of the total. If we can again assume that since Region 7 is one of ten regions, that 10% of those 250,000 orders in 1987 will be in Region 7, then 25,000 total orders will be selected for our report.

Looking back at Figure 6 at the calculations for an MR NOBUF read, we see that the retrieval will take 6,697 I/Os. These numbers do not change when we select a different number of records because we're not dealing with record pointers.

However, Figure 9 shows that our indexed retrieval is no longer our best choice. When we qualify this large a number of records, large blocked reads are our best choice.

CONCLUSION

The bottom line then is that once we understand and buy into the value of providing certain timely information to our users, there is no one answer or "black box" that we can plug into our application in order to optimize performance. There are a few basic rules, however that you can follow that will help guide you in your quest for providing information quickly:

1. Provide Management Reporting Systems using data bases designed specifically for retrieval purposes. Don't try to cut corners on disc space at the expense of timeliness of information.

2. Don't assume that using an IMAGE chain is always more efficient than a serial read. Depending on the number of entries in the chain and the blocking factor of the data set, even a standard IMAGE serial read may be more efficient.

3. Use tools such as Adager to reload detail data sets where you often do retrievals, particularly where chains are long and/or blocking factors high.

4. For on-line applications, ad-hoc reports and exception reporting, use sophisticated indexing techniques such as Omnidex to maximize performance.

5. For applications that extract large volumes of data, use a MR NOBUF tool such as Suprtool to minimize I/Os.

6. A combination of these techniques implemented intelligently can provide users with an extremely valuable informational system.

$$\frac{\text{250,000 record pointers for "87"}}{\text{64 record pointers per I/O}} = \text{3,907 I/Os}$$

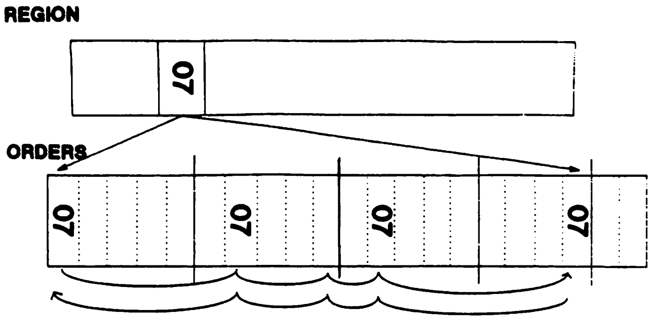
$$\frac{\text{75,000 record pointers for "07"}}{\text{64 record pointers per I/O}} = \text{1,172 I/Os}$$

5,079 I/Os

(to retrieve the records using the relative record numbers) 25,000 I/Os

30,079 I/Os

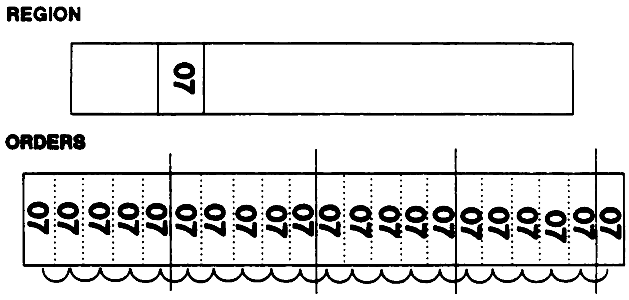
FIGURE 9



75,000 I/Os

FIGURE 4

AFTER SORTING



15,000 I/Os

FIGURE 5

CHAINED READ

75,000 RECORDS WITH REGION = "07"

75,000 I/Os

MR NOBUF READ

$$\frac{\text{28,732 BYTES/IO}}{\text{256 BYTES/RECORD}} = \text{112 RECORDS / I/O}$$

$$\frac{\text{750,000 RECORDS}}{\text{112 RECORDS/IO}} = \text{6,697 I/Os}$$

FIGURE 6

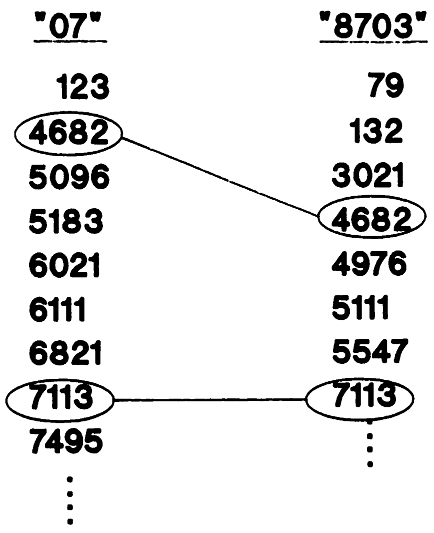

"07"
123
4682
5096
5183
6021
6111
6821
7113
7495
"8703"
79
132
3021
4682
4976
5111
5547
7113

FIGURE 7