

Understanding DataEase Indexes

Graham Smith
PLM Consulting, Inc.
gsmith@plmconsulting.com
July 2006

What is an Index?

If you are trying to find something in a book, you can either go through it page by page looking for what you want, or you can consult an index and see which page it is on. Now, not every book is going to have an index and not every word in a book is going to be in an index, but they can certainly be a help in a textbook.

A novel is a different situation, however. You would not expect to find an index in a novel because that's not how it is used. Likewise, in a text book, you would not expect to find the word 'and' in the index because no one is going to be turning to the index looking for that word in the first place. So, in creating an index for a book, you have to decide if there needs to be one and what needs to be in it.

In the case of a database, an index is much narrower. You can create an index on any field in the database. But the same rules apply, you don't want to bother with an index on something that is of no particular value. So, just how you decide what needs an index? For that matter, why not just index everything just in case? To understand the answer, it may help to understand just how an index works in DataEase.

How indexes work

Well, it's a lot more complicated than an index in a book, but for simplicity sake, we can think of the two as being somewhat alike. Let's say you have a table named OrderHeader and you frequently run reports that select all the orders placed on a single day or a range of dates. If you create an index on the OrderDate field, then DataEase will create a separate file (with an .i?? extension where ?? is the field number in hexadecimal - .i01 is an index on the first field, .i20 is an index on the 32nd field). The index will consist of an entry for every distinct (unique) date in the data, plus a set of 'pointers' to the records in the main data file that have that value. So, if there are 10 orders on 01/27/2006, then there will be an entry in the index for that date and it will have pointers to the 10 records in the table with that date.

Now, as I said, it is a lot more complicated than that and the structure of an index is somewhat different depending on what kind of data is being indexed, but the main point is to make it quicker to find the records you need to find. So, again the question becomes, "why not just index everything?"

The answer is fairly simple – every time you enter or update a record in a table, the indexes for that table also have to be updated. So that makes the question, "does it really

make sense to take the performance hit on the front end by indexing a given field?" If the answer is yes, then index it, otherwise, don't. There is also another downside to indexing that is rather more difficult to understand and that requires a certain amount of math to explain since it involves set theory and Cartesian products.

Lets say you have a table with 100,000 records in it and there are at least two indexes. Lets also say that you write a report that selects records based on these two indexes. Finally let's suppose that the data distribution is such that your selection on each index pulls up 30% of the records. The way this is going to work is that the report is going to find the set of the records that satisfy criteria 1 and the set that satisfies criteria 2, and is then going to find the intersection of the two sets to get the final set (the Cartesian product). This means that you have examined 60% of the records in the table. Where this gets really ugly is when you have several indexed fields in the selection and all of them represent a fairly large percentage of the total number of records. In a case like this, you can end up going through more than 100% of the records. This has come to be known as the Cartesian Catastrophe since it can end up taking far longer to select the records than if there were no indexes at all being used.

Now, I grant you that this sort of thing is unlikely to crop up that often but it is something to be aware of. It also points to a few rules of thumb in creating and using indexes:

- Don't put an index on a field unless you are sure it will help
- Testing never hurts
- Periodically review your index usage since changes in data distribution and what you are doing queries on is going to change
- When using indexes, always try and select the index that will give you the smallest record set first
- In general, a unique or key field should be indexed
- Don't bother with an index on a small table

Fast Text Indexes

In an application I am currently working on, there is one rather large table (currently about 580K records) that is going to continue to grow. Several years back, I did some tests in DFD to determine when and if a Fast-Text index would be of use. Since I am now working almost completely in DE65, I decided to repeat the tests with it and DE7.

For those not familiar with a Fast-Text index, it can be used with text and numeric strings and should provide improved performance with exact matches, particularly on longer text fields. FWIW, no index will help with a text field longer than 40 char since that is the max that either index will deal with. It should also be pointed out that a Fast-Index should be slower if you are either sorting or selecting by range. For that reason, there are often only

limited cases where this kind of index would help.

Also, you will note that I have said "should" rather than "will" here. This is because the math behind the design of the index says that this is what should happen. But as we all know, theory and practice are sometimes at odds with each other.

When I tested this in DFD, I found that there were actually cases where a Fast-Text index would be of advantage. But in doing the same tests in DataEase, I can find no advantage at all. I suspect that this is because the speed of the computers I am testing on now are so much greater than those I tested DFD on, that this speed trumps any speed advantage found in the index type itself. In fact, much to my surprise, I find that tests on a 40 char text field are actually marginally slower with a Fast-Text index. I say marginally because the speeds are such that the differences are negligible.

Suggestions for Indexing

Test, test, test. As noted above, computers and networks are much faster now and the speed advantages you gain from indexing are decreasing.

In general the first field on both sides of a Relationship should be indexed. The reason is simply that these are both key fields and key fields should generally be indexed.

In DFD, a DQL that does a count of tablename, will read the value from the RDRR file and is almost instantaneous. In DataEase (both 6 and 7) it actually appears to count the records in the table and can be very slow in a large table (my test file took about 6 minutes). However, if you precede the count with the command lock file tablename shared, the count is very fast (about 6 seconds in my test file).

Lastly, there has been a limit of 32K records to an indexed search at least as far back as DE45. If the search hits that many records then there is a section of code in the program that tells it to abort the indexed search and do a record by record search (a table scan) instead. I first found this in a DFD5 app and this was not changed up thru DE65. However, the code was changed in DE7 and the same test that 5.5 minutes in DE65 only takes 1 minute in DE7 (selecting about 45K out of 500K records).