1 Database Speed

There are a few things that effect database speed: database design quality (are indexes in right places, are tables properly setup, etc.), how you query (or use) the database (do you have many linear type searches), database product (are you running Microsoft Access or Oracle?), and drive speed (are you on a laptop hard-drive or on a SCSI, with RAID 5).

Whenever setting up an application, you need to understand where the bottleneck will be, and hopefully this little guide will provide enough background to make better decisions.

Keep in mind that there are *many* exceptions to the 'rules'. For example, Microsoft Access running on a RAM drive will likely be *much* faster than Oracle running on RAID 5—depending on how much RAM (for caching) the system has.

1.1 Database Design Quality

Well, the first thing to ensure is that your database is properly normalized, and is in second or third normal form (you should strive for Boyce-Codd normal form).

Once the database is properly setup, you need to ensure that you have proper indexes on pretty much all foreign keys. Keys themselves should usually be of `INT` type, unless you have some really good reason not to use numbers. You should probably also put an index on anything that appears in the `WHERE` clauses in your queries.

Whenever putting an index on a string, unless it's a very short string, you only need to put an index on the prefix. This will save space and make updates faster; note that indexes are often cached, so the smaller the index, the more of it that can be kept in memory at any point in time.

If you're searching for strings, ensure that operators you use take advantage of the indexes. Avoid using expressions like:

```
SELECT * FROM PERSON WHERE NAME LIKE '%BOB%'
```

Expressions like the above don't make use of indexes, even if you have an index on `NAME` it's useless in that case. To make use of indexes, you can turn that into:

```
SELECT * FROM PERSON WHERE NAME LIKE 'BOB%'
```

This doesn't really do the same thing, but this query will generally be *much* faster than the pervious one. Sometimes you can twist your queries around the right away to use indexes, sometimes you can't. Just make sure you don't pass on those opportunities where you can.

1.1.1 Diminishing Returns

Once you've setup everything as described, you get to a point of diminishing returns. At that point, by adding an index, or trying to be clever with queries, will actually cause the database to get *slower*.

Most major databases (Oracle, MS SQL Server) do query optimization. The way this optimization works may or may not be intuitive—so being overly clever with queries is mostly discouraged[1].

[1]Ignore people who tell you to 'optimize' your queries—there is no such thing anymore!

For example, if you setup indexes in places where it makes sense, nearly all of your queries should be non-linear anyway—they should all be logarithmic time. To query a database of N records should take somewhere $O(\log N)$ time. Queries that require linear (or worse) time should be avoided like the plague!

This database optimization gets even more interesting: You can't reliably time queries! Databases depend on caching—so even if you could time queries, your results will be vastly different at different times, depending on cache. Also, databases like MS SQL Server actually learn from their optimization strategy results, and apply what's learned to figure queries. Let's say it applies strategy A to a query. Next time it might apply strategy B to that same query. It might then compare the results, and apply the winning strategy more often[2].

1.1.2 Perception

Very often, it is not the actual query speed that's important, but the perceived query speed. How quickly do you get results? Let us say you're searching for 1000 records: your application could query for all 1000, and then show all the results at once. On the other hand, your application might find the first result and display it, then second result, and display that, and so on and so on. When people see results, they get less frustrated that "nothing is happening" (or watching the hour-glass mouse cursor).

So given the same database with the same application, you can make one *seem* faster than the other by simply timing things right and not letting the user wait. This actually is probably more important than anything else: make it *seem* fast, and it will be fast.

1.1.3 Prediction

Often, the actual results don't have to be exact. When you search Google, you get a very fast response, and yet it says it might say it has 20,000,000 matches. How can that be? How did they do that this quickly? Well, the trick is that they didn't.

These types of numbers are derived from how quickly you found your results, and how many records you have. For example, if I have ten billion records in the database, and I found first 20 in ten milliseconds in the first ten thousand records... then (all things being equal), I'll find another 20 records in another ten milliseconds by searching the next ten thousand records; i.e.: with those numbers, I can 'estimate' that I have 20,000,000 records and that it will take me 2.7 hours to find them all.

Usually this type of thing gets much more complicated as indexes get involved, but you can still judge rather accurately by the percentage of the database you've searched.

1.2 Database Products

Another used-to-be major performance issue is the database product. Over the last few years, this is much less the case. Most modern databases have pretty much the same performance when it comes to pretty much most of the tasks, but some databases do better at one type of tasks than others, and so on.

[2] It will still apply the non-winning strategy once in a while—to avoid being stuck in a local maxima.

For example, Oracle is fast—yet you can setup MySQL to be faster. Microsoft SQL Server is probably faster than most 'powerful' databases on Windows—but that's because they use the operating system at their core; disk scheduling is very important in databases, and Microsoft SQL Server uses the Windows to do that—while Oracle uses their own. MySQL just ignores that whole issue and uses the operating system provided file-system.

I guess the point is that they're all equally 'fast' if you apply them correctly in the right environment.

Now, there may be a major difference in speed between a professional database (like Oracle, SQL Server) and something non-professional like Microsoft Access. But they're databases meant for different markets—and whatever one is doing[3] with Microsoft Access is unlikely to be done in Oracle, and vice versa.

1.3 Drive Speed

Drive speed is *very* important when it comes to databases. If you're running anything critical consider SCSI. Also consider RAID 5. These may be expensive, but when it comes to speed, they're top of the line.

Another interesting approach is setting up the journal (transaction journal) on a fast drive—possibly battery backed RAM drive. Usually you'd use the expensive SCSI drive for this (and dedicate it to that task). Most of the updates and record locking, etc., will see a *huge* speed increase if the journal is on a fast drive.

1.3.1 General Case

More often than not, you don't have many resources (or actual power) to setup the system the way you want it to be setup, so the general case when it comes to hardware is 'the company' buys a server box from Dell or HP, and that's it. If you're lucky, it will have SCSI drives and some RAID.

One very important point you should push for is RAM. Lots and lots and lots of RAM. Databases cache disk sectors, they cache indexes, etc., and if you have plenty of RAM, most of your queries won't even hit the disk to begin with. That's a difference of 1000x in speed. So lots of RAM is *very* important.

1.4 Distributed Databases

Often speed is achieved by distributing the database load to several different computers. In these cases, usually you have many *read* machines, and one *write* server. The read machines synchronize their records once in a while with the write server. All updates go to the write server, and most[4] queries go to the read servers. For a pretty cool example of this, look up Slashdot architecture on the `slashdot.org` website.

Often some other form of software is used to distribute the load throughout the system—so the database may be totally unaware of anything. The middleware comes in useful in these cases—and this is where J2EE shines. Look into `Jboss`, and into load balancing thingies.

[3]Oh, how I wish that was the case!

[4]Queries that can suffer to be a bit out of date.