**A Very Rough Overview of Costs**

Then, once you’re able to view an execution plan, one of the great things about it is that you’re able to see the cost of not only the entire execution plan, but each individual operation that makes up the plan – simply by mousing-over it – as shown below:

And, again, a key thing to call out here is that these costs (estimated or otherwise) are based on SQL Server’s knowledge of the size of your tables as well as the cardinality and distribution of your data. Or, in other words, these costs are based upon statistics about your data. They’re not, therefore, something ‘tangible’ like the number of milliseconds associated with an operation. As such, the best way to think of them is that lower numbers are better – unless you want to try and get into some of the nitty-gritty details about how these costs are calculated (which, again, is proprietary information or part of SQL Server’s ‘secret sauce’).

With that said, there’s still a way to ‘frame’ these costs – to provide an idea of what costs roughly mean in the ‘real’ world.

* **.003.** Costs of .003 are about as optimized as you’re going to get when interacting with the storage engine (executing some functions or operations can/will come in at cheaper costs, but I’m talking here about full-blown data-retrieval operations).
* **.03.** Obviously, costs of .03 are a full order of magnitude greater than something with a cost of .003 – but even these queries are typically going to be VERY efficient and quick – executing in less than a second in the vast majority of cases.
* **1.** Queries with a cost of 1 aren’t exactly ugly or painfull (necessarily) and will typically take a second or less to execute. They’re not burning up lots of resources, but they’re also typically not as optimized as they could be (or they are optimized – but they’re pulling back huge amounts of data or filtering against very large tables).
* **5.** Queries with a cost greater than 5, by default, will be executed with a parallel plan – meaning that SQL Server sees these queries as being large enough to throw multiple processors/cores/theads-of-execution at – in order to speed up execution. And, if you’ve got a web site that’s firing off a query with a cost of 5 or more per every page load, for example, you’ll probably notice that the page ‘feels’ a bit sluggish loading – maybe by a second or two – as compared to a page that would ‘spring up’ if it was running a query with a cost of, say, .2 or lower. So, in other words, queries up in this range start having a noticeable or appreciable ‘cost’.
* **20.** Queries in this range are TYPICALLY going to be something you can notice taking a second or so. (Though, on decent hardware, they can still end up being instantaneous as well – so even at this point, things still depend on a lot of factors).
* **200.** Queries with this kind of cost should really only be for larger reports and infrequently executed operations. Or, they might be serious candidates for the use of additional tuning and tweaking (in terms of code and/or indexes).
* **1000.** Queries up in this range are what DBAs start to lovingly call ‘queries from hell’ – though it’s possible to bump into queries with costs in the 10s of thousands or even more – depending upon the operations being executed and the amount of data being poured over.

And, in case it’s not obvious from some of the descriptions above, the ‘thresholds’ I’ve outlined above REALLY need to be taken with a grain of salt – meaning that they’re just rough approximations to try and give these numbers a bit of context (for those that aren’t very experienced with performance tuning in general).

**The True Cost of Operations**

However, while taking a single query and comparing its cost in isolation is a great way to tune that operation to get better performance out of it (i.e., by adding/tuning indexes and/or tuning the code to make it better your GOAL is to decrease costs – since LOWER costs are BETTER costs), it isn’t a viable way to know what your most expensive query on a given server or within a given database is.

For example, which query is truly an ‘uglier’ query from a performance standpoint? That big/horrible/ugly report that management likes to run once a day at 7PM with a total cost of 1820.5? Or a single operation with a cost of .23 that gets called over 800,000 times in the same day? Typically a query with a cost of .23 won’t really be scrutinized that much – because it’s ‘optimized enough’. But if it’s called at highly repetitive rates, then that cost is incurred over and over and over again – typically during periods of peak load on your server. And, in fact, if you multiply .23 \* 800K, you end up with a total, aggregate, cost of 184,000 – something that makes that ‘nightly’ query look like child’s play.

As such, finding your most expensive queries is really a question of juggling execution costs against execution counts – because it’s only when you consider both concerns that you start getting a sense for the TRUE costs of your most expensive operations.