DIRECCION:

https://thomaslarock.com/2013/05/top-3-performance-killers-for-linked-server-queries/

14 MAY TOP 3 PERFORMANCE KILLERS FOR LINKED SERVER QUERIES

Posted at 13:48h in [**MSSQL**](https://thomaslarock.com/category/mssql/), [**SQL MVP**](https://thomaslarock.com/category/sql-mvp/), [**SQL Server Performance**](https://thomaslarock.com/category/sql-server-performance/), **[SQLServerPedia Wiki](https://thomaslarock.com/category/sqlserverpedia-wiki/)** by [**Thomas LaRock**](https://thomaslarock.com/author/sqlrockstar/) [**54 Comments**](https://thomaslarock.com/2013/05/top-3-performance-killers-for-linked-server-queries/#disqus_thread)

[**Share**](javascript:void(0))

One thing I have noticed in all my years as a data professional: few users understand (or care) how far away they are from their data. Quite often they expect instant results from their queries despite there currently being an upper bound due to things like network bandwidth, the speed of light, and the data sitting on a server on the other side of the world.

SQL Server makes it easy to connect to and query data from remote data sources. The common way of getting this done is through the use of a [**linked server**](http://msdn.microsoft.com/en-us/library/ms188279.aspx), which is little more than an OLEDB data source.

The beauty of a linked server connection is that it allows an end user to write a query that looks like any other T-SQL query. For example, here is a query against a local table:

|  |
| --- |
| SELECT col1, col2 FROM [databasename].[schemaname].[tablename] |

And here is a query that would be written that would utilize a linked server:

|  |
| --- |
| SELECT col1, col2 FROM [linkedservername].[databasename].[schemaname].[tablename] |

To someone with an untrained eye they would think the data is as easily accessible as any other. You don’t have to do anything special to write a query against a remote data source other than include the name of the linked server.

But the dirty little secret here is that SQL Server is going to make decisions for you as to how to mash all the data together and return you a result set.

People love using linked servers. Because data can (and does) exist everywhere, users naturally want to write one query that joins as much data as possible with no regard if it is local or remote. Even a DBA with many servers to manage will be tempted to build out a series of linked servers in order to capture monitoring details in a central location.

There is also the case where data is going to be too big or cumbersome to move around easily. In that case you are going to want the query to be executed on the remote server and only return the data that is needed across the network. What this means that you should make an effort to help SQL Server make the right choices.

Here are the top three issues I have seen when it comes to linked server (AKA Distributed Query) performance:

1. INSUFFICIENT PERMISSIONS

Without a doubt this is the number one reason for why linked server query performance suffers. Historically in order for SQL Server to take advantage of using statistics on the remote server then the login used to make the connection on the remote servers needed sufficient rights. The role needed would have been one of the following:

* sysadmin
* db\_owner
* db\_ddladmin

If you don’t have sufficient permissions then you aren’t able to use stats, and this is killing your performance across that linked server connections. So for everyone that has been assigning the db\_datareader role to remote logins you are sacrificing performance for security. While that may be an acceptable tradeoff in your shop, I am willing to wager that most admins have no idea about this silent performance killer.

A good example of identifying these symptoms are contained in this article: [**http://www.sql-server-performance.com/2006/api-server-cursors/**](http://www.sql-server-performance.com/2006/api-server-cursors/)

In SQL 2012 SP1 the permissions to view the statistics on an object have been modified so that a user with SELECT permission would be able to use the stats on the remote tables. [**Check this link for more details in the ‘Permissions’ section towards the bottom**](http://technet.microsoft.com/en-us/library/ms174384.aspx).

2. QUERY JOIN SYNTAX

[**Conor Cunningham gave a great talk at SQLBits X on Distributed Queries**](http://sqlbits.com/Sessions/Event10/Distributed_Query_Deep_Dive). In that talk he discussed some different join types and whether or not they were good candidates to be executed remotely. For example, a simple query that wants to pull data from just one remote table is likely to be executed remotely and only pull back the necessary rows. But what about a join between one small local table and one large remote table? What happens then?

You’ll need to watch the video to listen to Conor explain all the nuances of distributed queries. The lesson I learned from the talk is simple: when possible, give the optimizer some help. If you can rewrite your query to only pull back the smallest rowset possible from the remote server, the better chance you have of that query being executed remotely.

3. FUNCTIONS

Some functions, like GETDATE(), won’t be executed remotely. This make complete sense to me because if I am trying to execute (and filter) a set of data on a remote server (where the time could be very different than the local server) then the concept of GETDATE is lost due to the fact that the servers could be in multiple timezones. SQL Server knows this and as a result it likely won’t execute the query remotely.

OK, let’s say you have identified a linked server query that is not performing well. What can you do?

The way I see it, you have two options available for helping to tune a linked server query.

The first option is to **force the query to run remotely**. This is done by using the [**OPENQUERY() function**](http://msdn.microsoft.com/en-us/library/ms188427(v=sql.110).aspx). This function will force the query to be run on the remote server and by default will use the permissions defined in the linked server. We’ve already talked about the potential issue with permissions up above, so I won’t do that again here.

The second option is to try **rewriting the query**. (I know many developers that just screamed at me to “STOP BLAMING THE CODE!”) Start with one table and the necessary predicates and add in additional tables and predicates until you find the one that changes the results from being executed remotely into ones that are done locally. That will help you identify which part(s) of the query are making SQL Server to switch to running locally. Once identified you can then go about exploring some rewrite options.

I’ve been writing queries for linked servers for about a dozen years now. These are the top three performance killers I have found to be common to many shops. Microsoft has done well to help remedy the permissions issue. However they are not so good as to write your queries for you (at least not yet). Until that day happens you are going to want to spend the extra time testing your queries to make certain that they are behaving as expected.