UPDATE STATISTICS: the Secret IO Explosion

January 29, 2014 Kendra Little Index Maintenance, SQL Server, Statistics
84 Comments

I first knew something was up when I looked at the job history for a simple maintenance plan. It had two steps:

1. Rebuild all the indexes in the database – this took 10 minutes each night.
2. Update statistics – this took 2-3 hours each night.

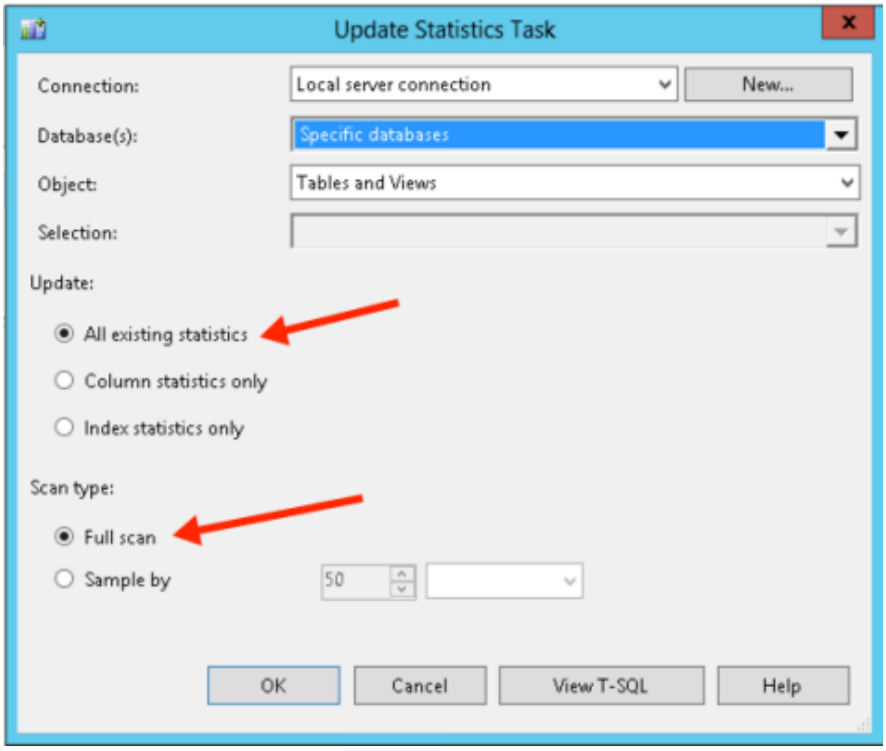
What was going on? Statistics in SQL Server are small, lightweight objects. Indexes are larger and contain more data. Why would updating statistics take so much longer?

Maintenance Plans light the fuse

I love the concept of maintenance plans, but I don't love the way all the tasks are set up.

In the case I was looking at, the Update Statistics task was being used with two values that are set by default:

- Run against all statistics
- Update them with fullscan



"All" statistics means that both "column" and "index" statistics will be updated. There may be quite a lot of statistics — most people leave the "auto create statistics" option enabled on their databases, which means that queries will dynamically cause the creation of more and more statistics over time.

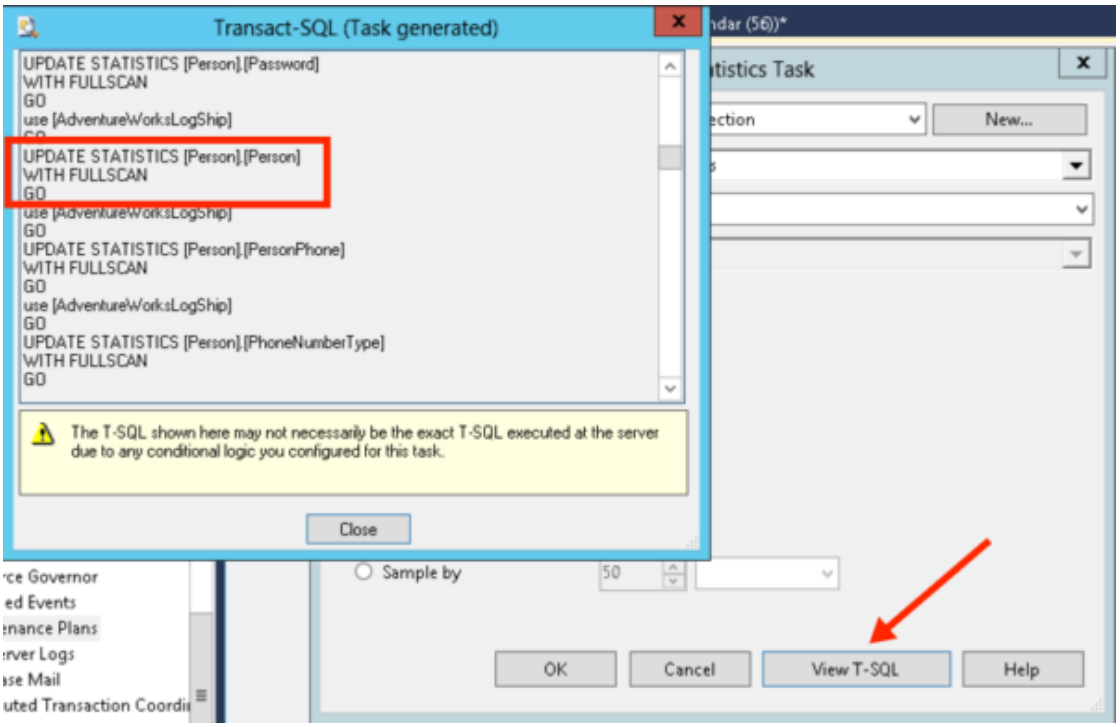
Combined with "fullscan", updating all statistics can become a significant amount of work. "Fullscan" means that to update a statistic, SQL Server will scan 100% of the values in the index or column. That adds up to a *lot* of IO.

Why 'SELECT StatMan' repeatedly scans tables

If SQL Server needs to update column level statistics for the same table, it could potentially use a single scan and update multiple stats, right?

Because of the runtimes I was seeing, I was pretty sure that wasn't happening. But we can take a closer look and see for ourselves.

In our maintenance plan task, if we hit "View TSQL", a window pops up showing us the comamnds that the plan will run. (I love this feature, by the way!) We will use one of these commands to test things out in a bit.



First, let's make sure we have some column level statistics on our database. It already has indexes and their associated stats. To create some column level stats, I run these queries:

```
--create two column stats using 'auto create statistics'
select * from Person.Person where MiddleName like 'M%';
select * from Person.Person where Title is not null;
GO

--Create two filtered stats on Title
create statistics kl_statstest1 on Person.Person (Title) where Title = 'Mr.'
GO
create statistics kl_statstest2 on Person.Person (Title) where Title = 'Ms.'
GO
```

That will create two "auto" stats what start with "_WA_Sys", and two stats that I named myself. To check 'em out and see ALL the index and column stats on the table, we run:

```
exec sp_helpstats 'Person.Person', 'All';
GO
```

Results | Messages

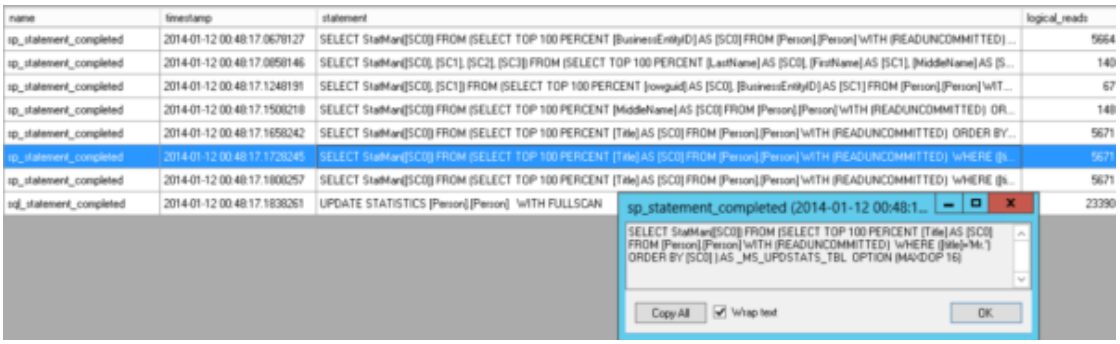
	statistics_name	statistics_keys
1	_WA_Sys_00000004_693CA210	Title
2	_WA_Sys_00000006_693CA210	MiddleName
3	AK_Person_rowguid	rowguid
4	IX_Person_LastName_FirstName_MiddleName	LastName, FirstName, MiddleName
5	kl_statstest1	Title
6	kl_statstest2	Title
7	PK_Person_BusinessEntityID	BusinessEntityID

Sure enough, this shows us that we have seven stats total– three are related to indexes.

Alright, time to run that sample command excerpted from our maintenance plan. I start up an Extended Events trace to capture IO from sp_statements completed, then run the command the maintenance plan was going to use to update every statistic on this table with fullscan:

```
UPDATE STATISTICS [Person].[Person]
WITH FULLSCAN
GO
```

Here's the trace output –click to view it in a larger image:



name	timestamp	statement	logical_reads
sp_statement_completed	2014-01-12 00:48:17.0678127	SELECT StatMan([SC0]) FROM (SELECT TOP 100 PERCENT [BusinessEntityID] AS [SC0] FROM [Person].[Person] WITH (READUNCOMMITTED) ...	5664
sp_statement_completed	2014-01-12 00:48:17.0858146	SELECT StatMan([SC0], [SC1], [SC2], [SC3]) FROM (SELECT TOP 100 PERCENT [LastName] AS [SC0], [FirstName] AS [SC1], [MiddleName] AS [S...	140
sp_statement_completed	2014-01-12 00:48:17.1248191	SELECT StatMan([SC0], [SC1]) FROM (SELECT TOP 100 PERCENT [rowguid] AS [SC0], [BusinessEntityID] AS [SC1] FROM [Person].[Person] WIT...	67
sp_statement_completed	2014-01-12 00:48:17.1508218	SELECT StatMan([SC0]) FROM (SELECT TOP 100 PERCENT [MiddleName] AS [SC0] FROM [Person].[Person] WITH (READUNCOMMITTED) OR...	148
sp_statement_completed	2014-01-12 00:48:17.1658242	SELECT StatMan([SC0]) FROM (SELECT TOP 100 PERCENT [Title] AS [SC0] FROM [Person].[Person] WITH (READUNCOMMITTED) ORDER BY...	5671
sp_statement_completed	2014-01-12 00:48:17.1728245	SELECT StatMan([SC0]) FROM (SELECT TOP 100 PERCENT [Title] AS [SC0] FROM [Person].[Person] WITH (READUNCOMMITTED) WHERE ([ti...	5671
sp_statement_completed	2014-01-12 00:48:17.1808257	SELECT StatMan([SC0]) FROM (SELECT TOP 100 PERCENT [Title] AS [SC0] FROM [Person].[Person] WITH (READUNCOMMITTED) WHERE ([ti...	5671
sql_statement_completed	2014-01-12 00:48:17.1838261	UPDATE STATISTICS [Person].[Person] WITH FULLSCAN	23390

Looking at the Extended Events trace output, I can see the commands that were run as well as their logical reads. The commands look like this:

```
SELECT StatMan([SC0]) FROM
(SELECT TOP 100 PERCENT [Title] AS [SC0]
FROM [Person].[Person] WITH (READUNCOMMITTED)
WHERE ([title]='Mr.') ORDER BY [SC0] )
AS _MS_UPDSTATS_TBL  OPTION (MAXDOP 16)
```

The "logical_reads" column lets me know that updating four of these statistics had to do four separate scans of my table– and three of them are all on the Title column! (Doing a SELECT * FROM Person.Person shows 5,664 logical reads by comparison.)

IO was lower for statistics related to nonclustered indexes because those NC indexes have fewer pages than the clustered index.

A better way to update statistics: Let SQL Server pick the TABLESAMPLE

If you just run the TSQL command 'UPDATE STATISTICS Person.Person' (without telling it to scan all the rows), it has the option to do something like this:

```
SELECT StatMan([SC0], [SB0000])
FROM
(SELECT TOP 100 PERCENT [SC0],
    step_direction([SC0]) over (order by NULL) AS [SB0000]
    FROM
     (SELECT [Title] AS [SC0]
     FROM [Person].[Person] TABLESAMPLE SYSTEM  (3.547531e+001 PERCENT) WITH (RE
OPTION (MAXDOP 1)
```

It dynamically figures out a sample size by which to calculate results! (It can pick a variety of options– including scanning the whole thing.)

How to configure faster, better statistics maintenance

Avoid falling for the pre-populated settings in the "Update Statistics" task in the maintenance plan. It's rare to truly need to use FULLSCAN to update stats in SQL Server, and even when cases where it's justified you want to implement that with statements targeting the individual statistics to update. The basic "UPDATE STATISTICS Schema.TableName" command is pretty clever– the issue is simply that Maintenance Plans don't make it easy for you to run that!

Unfortunately, if you use maintenance plans there's no super simple solution– it forces you to specify either fullscan or a specific sample. There's no way to just use the basic "You compute the minimum sample" with that task.

You've still got good options, they're just a few more steps:

- You could use a t-sql related task or a custom SQL Agent job to run sp_updatestats
- You could use a free index and statistics maintenance script. The example I've linked to is super clever, and avoids updating statistics where it has just rebuilt an index!
- You could also let auto update stats take care of the issue– that's often just fine on small databases or where there aren't major data fluctuations

And each of those options should chew up less IO than updating all index and column statistics with FULLSCAN.

Kendra Little

My goal is for you to understand your SQL Server's behavior– and learn how to change it. When I'm not figuring out the solutions to your database problems, you'll find me at user group meetings in Portland, Oregon. I also love to draw.

Previous Post
Reporting in Production: SQL Server (video)

Next Post
What Do You Ask the Leaving DBA?

84 Comments. Leave new

Allen McGuire January 29, 2014 1:15 pm

I actually had a strange performance issue result from stats being updated this morning – a job went from 44 minutes right after the update took place. First thought: developers deployed something 😉 After a little analysis and a new NC index, all was well. After that, however, I still wanted to find out what happened so I turned to my maintenance jobs.

To investigate my index/stats maintenance history, I have a report I created to query Ola's logging table if anyone is interested. Turns out the table I put an index on had it's stats updated, obviously throwing the execution plan out of whack. You would have to update the data sources and parameter defaults, but that's about it.

http://allen-mcguire.blogspot.com/2014/01/rdl-for-olas-index-maintenance-logging.html

Reply

Allen McGuire January 29, 2014 1:16 pm

Ah – posts don't like greater than/less than signs – job went from under a minute to over 44 minutes 😉 Some of my words got hacked out.

Reply

Chris Woods August 28, 2014 5:13 pm

Allen–
That link no longer goes to the article on your website. Do you still happen to have the post?

CW

Reply

Kendra Little August 28, 2014 5:17 pm

The URL looks like it's just been updated. Here's an updated link, or you can search for the term RDL on his blog in the search box at the top right.

Reply

Klaas January 30, 2014 2:20 am

Thanks Kendra

So would you recommend to SET AUTO_CREATE_STATISTICS OFF ?

Reply

Kendra Little January 30, 2014 10:37 am

Hi Klaas,

I recommend leaving that ON — statistics are extremely helpful to the optimizer for query plan quality.

I just wouldn't update all your statistics with fullscan. The statistics are good and lightweight, but the maintenance plan task isn't great.

Hope this helps!
Kendra

Reply

Klaas January 31, 2014 2:34 am

OK