

# Workflow for query tuning

Last updated by | Ricardo Marques | Mar 15, 2023 at 3:37 AM PDT

## Contents

- [1 Making the right questions](#)
- [2 The problem is with a specific query](#)
  - [21 plan regression parameter sniffing and recompile](#)
  - [22 Update statistics](#)
  - [23 plan compilation timeout](#)
  - [24 implicit conversions and non-SARGable predicates](#)
  - [25 missing index](#)
  - [26 Wait stats](#)
  - [27 Operators](#)
  - [28 query logic](#)
- [3 Issue in general](#)
- [4. Customer claims that a query returns the incorrect num...](#)

The following TSG is design to help you with query tuning troubleshooting. Start at point [1](#)

## 1 Making the right questions

First scope the problem. Is the customer asking for help to tune a specific query or is he facing problems in general?

If the customer is complaining with query execution, not related with errors or performance, but with the number of rows that are being returned, go to [4](#).

In some cases the customer might reference "a Job" or "when we open a dashboard on the application", but without knowing what is the query (or multiple queries). If you are able to get those queries, you can jump into an approach on where you can troubleshoot from a specific query point of view (even if you have more than one).

The customer can use [SQL Profiler](#) or [Extended Events](#) to find queries that are taking more time.

To troubleshoot from a specific query perspective proceed to [2](#)

If you dont have a set of queries, go to [3](#)

## 2 The problem is with a specific query

With a specific query in hand, there are two very important components that you must have always:

- the query [execution plan](#). Check the TSG on how to get it. Note that the execution plan (and query execution time) that is retrieved from SSMS might differ from what is present on QDS (execution plan from

when the query is executed from the application). Check the TSG [Different execution plan when executing from SSMS](#)

- the wait types associated with query (if you have the actual execution plan, it will contain the wait types). IF can associate the query with the query hash, you can see on [telemetry](#).

The analysis of an execution plan can be complex and some points might not be covered here. But here we have the most common scenarios and path that you can follow when analyzing an execution plan.

The customer mentioned that the query was running faster before? If yes go to [21](#)

If you have the actual execution plan, the first thing that you want to check is the ratio between Actual and Estimated. Like mentioned before, this elements can be checked on the actual [execution plan](#).

Do you see a huge gap between actual and estimated rows? Or do you see high memory grants? If yes, proceed go to [22](#)

Look at the top of the execution plan and check the value for StatementOptmLevel. Is it TRIVIAL, FULL or TIMEOUT? If it is timeout, go to [23](#)

Is there any implicit conversion on the plan. If yes, proceed to [24](#)

Now lets look at the most obvious. Are there any missing indexes? If yes go to [25](#)

If they are available, what are the wait stats for the query? Jump to [26](#) and lets take a look at them

Now looking at each operator. Look at the percentages. Check the operators with high percentages. Go to [27](#) and lets check them

Now lets look at the query logic. Jump to [28](#)

## 21 plan regression parameter sniffing and recompile

On this cases, something might have changed:

1- table size is now bigger and now the query requires tuning 2- the plan regressed 3- parameter sniffing 4- or the the query recompiled

For the points 2 and 3 you can check the TSG for [plan regression](#) and for [parameter sniffing](#). For point 4, check if the query recompiled or compiled a new plan by running [this](#) Kusto query. Just perform the search using the query hash. Could be that the plan changed due to statistics (so you have to update statistics) or due to schema change (there is an index that is missing). For point 1, just proceed to the next points.


## 22 Update statistics

Look at the plan and identify the tables involved on the query. Update the statistics with FULLSCAN for those tables. For example: `UPDATE STATISTICS [dbo].[Table] WITH FULLSCAN`

Run the query again - it should compile a new plan.

Note: it might be frustrating for a customer if we ask to update the statistics for all tables (depending on the database size, might take long). Just focus on the tables involved on the query that you are troubleshooting.

## 23 plan compilation timeout

This can happen due to query complexity or due to available resources during plan compilation. Check [this document](#) 

## 24 implicit conversions and non-SARGable predicates

If you see implicit conversions on the plan we have to fix them. Check [this TSG](#) for solutions.

Check also if the query contains [non-SARGable predicates](#) that might cause problems on data access.

## 25 missing index

You might wonder why you only check the missing indexes at this point. The reason is, because you want to make sure that you have updated statistics and make sure that you don't have implicit conversions on the plan. For example, on a plan where implicit conversions are present, the missing index suggested suggest you to put that column as included column. In theory would make sense, since there is an implicit conversion, but it would make more sense to fix the implicit conversion and create the index with that column as key, and not as included column.

Be aware that SSMS graphical interface doesn't show all missing indexes, like described [here](#). Also be aware of the limitations of missing indexes and rules on indexes design - [Indexing guidelines](#)

## 26 Wait stats

Look at plan waits using has reference [this TSG](#)

## 27 Operators

You can use [this TSG](#) as reference when analyzing execution plan operators. Things to look at:

- given the number of rows, a certain join algorithm would not make sense
- Big sort operations (if this is the case, check if the data can be sorted on a index)

Remember also that not always the SQL optimizer suggests indexes. We might can come up with an index by looking at the query. Look at this [TSG](#) on how to design an index.

## 28 query logic

This is by far the most complex point and in some cases might require knowledge of the data (that we don't have has support engineers). But some points to look at:

This [TSG](#) offers you some points on where to look at.

Also look at some obvious points like:

- row by row operations
- repeated calls to a subquery or CTE. Use temp tables instead.

## 3 Issue in general

Generate an ASC report. Go to **Performance** -> **Overview** and look at the resource metrics. Is there an noticeable metric?

If you see High CPU use this [workflow](#). If you see High IO is this [workflow](#). If you don't see a specific metric with an high value, proceed.

Now we want to know exactly what is going on at the instance level, more specifically on what are queries waiting for. Like so, on **Performance->Overview**, check what are the most relevant wait types.

You can also use the query provided on point 2 from this [TSG](#) to get all the processes that are currently running. From the output you can have some clarity on:

- what queries are running
- what are the wait types
- what is the cached execution plan

From here you can tackle the problem by:

- checking the problem from a [Wait Type](#) perspective, even though that in some cases you might want to find the specific queries and addressing them correctly. Sometimes a problem that is general can be found in a set of queries. Take for example an application where all the variables are declared in way that implicit conversions will occur.
- using the wait type to narrow the problem to a set of queries that have high values for that same wait. After finding those queries, you can than start looking at specific queries using the step [2](#). The queries can be found by using this [query](#) referenced on point 2, where you can get the queries if they are occurring now, or you can use this [Kusto query](#). Just change the value for the relevant wait type. on ASC, going to **Performance -> Queries -> Top Waiting Queries** you might also get the relevant query hashes.

#### 4. Customer claims that a query returns the incorrect number of rows

The issue will be related with query code and logic, that are owned by the customer. So it will be out of our scope. Anyway we can give some guidelines on how the customer can troubleshoot. Check [Debug T-SQL execution methods - guidelines to troubleshoot row counts expected](#)

**How good have you found this content?**

