What steps should be necessary to optimize a poorly performing query?

Asked 16 years, 3 months ago Modified 7 years, 5 months ago

Viewed 6k times

10

I know this is a broad question, but I've inherited several poor performers and need to optimize them badly. I was wondering what are the most common steps involved to optimize. So, what steps do some of you guys take when faced with the same situation?

> Related Question:
> [What generic techniques can be applied to optimize SQL queries?]

sql-server optimization

Share

Improve this question

Follow

edited May 23, 2017 at 12:08

Community Bot

1 • 1

asked Sep 14, 2008 at 0:13

Kilhoffer

32.9k • 22 • 99 • 124

Looks very similar to this older question. – Unsliced Sep 14, 2008 at 7:23

7 Answers

Sorted by: Highest score (default)

16

1. Look at the execution plan in query analyzer
2. See what step costs the most
3. Optimize the step!
4. Return to step 1 [thx to **Vinko**]

Share Improve this answer

Follow

edited Jul 4, 2017 at 10:18

Alireza

105k ● 27 ● 277 ● 173

answered Sep 14, 2008 at 0:24

roman m

26.6k ● 31 ● 104 ● 136

1 This is a loop and looping is bad in sql. :) I kid.. I kid.. Love this answer. – Joshua Hudson Sep 14, 2008 at 2:52

If you need more details, you can also examine trace output (use `SET AUTOTRACE ON` in Oracle). This will let you see things like physical reads, etc. I'd only use it after following roman's advice, though. – Hank Gay Sep 14, 2008 at 13:02

this answer assumes that the query itself cannot be rewritten to be more efficient. I always look at the query first and write it 4 or 5 ways and to see if any have an immediate benefit. – Mark Brady Oct 24, 2008 at 20:16

In SQL Server you can look at the Query Plan in Query Analyzer or Management Studio. This will tell you the rough percentage of time spent in each batch of statements. You'll want to look for the following:

- Table scans; this means you are completely missing indexes
- Index scans; your query may not be using the correct indexes
- The thickness of the arrows between each step in a query tells you how many rows are being produced by that step, very thick arrows means you are processing a lot of rows, and can indicate that some joins need to be optimized.

Some other general tips:

- A large amount of conditional statements, such as multiple if-else statements, can cause SQL Server to constantly rebuild the query plan. You can check for this using Profiler.
- Make sure that different queries aren't blocking each other, such as an update statement blocking a select statement. This can be avoided by specifying the (nolock) hint in SQL Server select statements.
- As others have mentioned, try out the Performance Tuning wizard in Management Studio.

Finally, I would highly recommend creating a set of load tests (using Visual Studio 2008 Test Edition), which you

can use to simulate your application's behavior when dealing with a large amount of requests. Some SQL performance bottlenecks only manifest themselves under these circumstances, and being able to reproduce them makes it a lot easier to fix.

Share Improve this answer

Follow

answered Sep 14, 2008 at 2:43

tbreffni

5,132 ● 5 ● 32 ● 30

3

Indexes may be a good place to start...

The low hanging fruit can be knocked down with the **SQL Server** Index **Tuning Wizard**.

Share Improve this answer

Follow

edited Jul 4, 2017 at 10:18

Alireza

105k ● 27 ● 277 ● 173

answered Sep 14, 2008 at 0:24

Andy S

8,861 ● 6 ● 38 ● 40

> I agree that this can be a good place to start, but after a while I've found that it gets a little conflicting with itself. It wants to remove indexes it has created for a previous tuning attempt. – Scott Bennett-McLeish Sep 14, 2008 at 4:09

I'm not sure about other databases, but for SQL Server I recommend the Execution Plan. It very clearly (albeit with lots of vertical and horizontal scrolling, unless you've got

2

a 400" monitor!) shows what steps of your query are sucking up the time.

If you've got one step that takes a crazy 80%, then maybe an index could be added, then after tweaking the index, re-run the Execution Plan to find your next biggest step.

After a couple tweaks you may find that there really are no steps that stand out from the others i.e. they're all 1-2% each. If that is the case, then you might then need to see if there is a way you can cut down the amount of data included in your query, do those four million closed sales orders need to be included in the "Active Sales Orders" query? No, so exclude all those with STATUS='C' ... or something like that.

Another improvement you'll see from the Execution Plan is bookmark lookups, basically it finds a match in the index, but then SQL Server has to quickly trawl through the table to find the record you want. This operation might at times take longer than just scanning the table in the first place would have, if that is the case, do you really need that index?

With indexes, and especially with SQL Server 2005 you should look to the INCLUDE clause, this basically allows you to have a column in an index without really being in the index, so if all the data you need for your query is in your index or is an included columnn then SQL Server doesn't have to even look at the table, a big performance pickup.

Share Improve this answer Follow

edited Sep 14, 2008 at 5:55

answered Sep 14, 2008 at 4:07

Scott Bennett-McLeish

9,277 ● 11 ● 42 ● 47

2

There are a couple of things you can look at to optimize your query performance.

1. Ensure that you just have the minimum of data. Make sure you select only the columns you need. Reduce field sizes to a minimum.
2. Consider de-normalising your database to reduce joins
3. Avoid loops (i.e. fetch cursors), stick to set operations.
4. Implement the query as a stored procedure as this is pre-compiled and will execute faster.
5. Make sure that you have the correct indexes set up. If your database is used mostly for searching then consider more indexes.
6. Use the execution plan to see how the processing is done. What you want to avoid is a table scan as this is costly.
7. Make sure that the Auto Statistics is set to on. SQL needs this to help decide the optimal execution. See

Mike Gunderloy's great post for more info. Basics of Statistics in SQL Server 2005

8. Make sure your indexes are not fragmented Reducing SQL Server Index Fragmentation
9. Make sure your tables are not fragmented. How to Detect Table Fragmentation in SQL Server 2000 and 2005

Share Improve this answer Follow

edited Sep 14, 2008 at 15:43

answered Sep 14, 2008 at 15:37

Leo Moore
2,158 ● 2 ● 19 ● 22

By all means, re-think your entire database architecture to tune a single query. De-normalizing isn't a query tuning step. Table scans aren't costly when the other choice is performing an index scan followed by table access by rowid when you need 100% of the rows in the first place. – Mark Brady Oct 24, 2008 at 20:25

Maybe the problem is the database architecture.
– Leo Moore Feb 23, 2012 at 11:25

1

The execution plan is a great start and will help you figure out what part of your query you need to tackle.

Once you figure out the where, it is time to tackle the how and why. Take a look at the type of queries you are trying to preform. Avoid loops at all cost as they are slow. Avoid

cursors at all costs because they are slow. Stick to set based queries when ever possible.

There are ways to give sql hints on the type of joins to use if you are using joins. Be careful here though, while one hint may speed up your query once, it may slow down your query 10 fold the next time through depending on the data and parameters.

Finally, make sure your database is well indexed. A good place to start is any field that is contained in a where clause probably should have a index on it.

Share Improve this answer

Follow

answered Sep 14, 2008 at 2:45

Joshua Hudson

2,217 ● 2 ● 21 ● 24

1

Look at the indexes on the tables that make the query. An indexes may be needed on particular fields that participate in the where clause. Also look at the fields used in the joins in the query (if joins exist). If indexes already exist, look at the type of index.

Failing that (because there are negatives to using locking hints) Look at locking hints and explicitly naming the index to use in the join. Using NOLOCKS is more obvious if you're getting a lot of deadlocked transactions.

Do what roman and Andy S mentioned first though.

Share Improve this answer

edited Sep 14, 2008 at 3:46

Follow

answered Sep 14, 2008 at 0:38

Rob Gray
3,266 4 36 34