Favourite performance tuning tricks [closed]

Asked 16 years, 4 months ago Modified 2 years, 3 months ago

Viewed 40k times

129

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 13 years ago.

When you have a query or stored procedure that needs performance tuning, what are some of the first things you try?

sql sql-server database performance

Share

Improve this question

Follow

edited Aug 27, 2014 at 11:44

community wiki

3 revs, 3 users 100%

Terrapin

Here are some SQL Server Query Optimization tricks – SQLMenace Sep 22, 2008 at 14:14

I agree that this is not constructive and can be searched in Google, but why it has 118 uv?! :) – FLICKER May 17, 2016 at 20:36

29 Answers

Sorted by: Highest score (default)

115

Here is the handy-dandy list of things I always give to someone asking me about optimisation.
We mainly use Sybase, but most of the advice will apply across the board.

SQL Server, for example, comes with a host of performance monitoring / tuning bits, but if you don't have anything like that (and maybe even if you do) then I would consider the following...

99% of problems I have seen are caused by putting **too many tables in a join**. The fix for this is to do half the join (with some of the tables) and cache the results in a temporary table. Then do the rest of the query joining on that temporary table.

Query Optimisation Checklist

- Run UPDATE STATISTICS on the underlying tables

    - Many systems run this as a scheduled weekly job
- Delete records from underlying tables (possibly archive the deleted records)
    - Consider doing this automatically once a day or once a week.
- Rebuild Indexes
- Rebuild Tables (bcp data out/in)
- Dump / Reload the database (drastic, but might fix corruption)
- Build new, more appropriate index
- Run DBCC to see if there is possible corruption in the database
- Locks / Deadlocks
    - Ensure no other processes running in database
        - Especially DBCC
    - Are you using row or page level locking?
    - Lock the tables exclusively before starting the query
    - Check that all processes are accessing tables in the same order
- Are indices being used appropriately?
    - Joins will only use index if both expressions are exactly the same data type

    - Index will only be used if the first field(s) on the index are matched in the query
    - Are clustered indices used where appropriate?
        - range data
        - WHERE field between value1 and value2
- Small Joins are Nice Joins
    - By default the optimiser will only consider the tables 4 at a time.
    - This means that in joins with more than 4 tables, it has a good chance of choosing a non-optimal query plan
- Break up the Join
    - Can you break up the join?
    - Pre-select foreign keys into a temporary table
    - Do half the join and put results in a temporary table
- Are you using the right kind of temporary table?
    - `#temp` tables may perform much better than `@table` variables with large volumes (thousands of rows).
- Maintain Summary Tables
    - Build with triggers on the underlying tables
    - Build daily / hourly / etc.
    - Build ad-hoc

    - Build incrementally or teardown / rebuild
- See what the query plan is with SET SHOWPLAN ON
- See what's actually happenning with SET STATS IO ON
- Force an index using the pragma: (index: myindex)
- Force the table order using SET FORCEPLAN ON
- Parameter Sniffing:
    - Break Stored Procedure into 2
    - call proc2 from proc1
    - allows optimiser to choose index in proc2 if @parameter has been changed by proc1
- Can you improve your hardware?
- What time are you running? Is there a quieter time?
- Is Replication Server (or other non-stop process) running? Can you suspend it? Run it eg. hourly?

Share Improve this answer Follow

edited Jan 22, 2015 at 15:39

community wiki
9 revs, 3 users 97%
AJ.

---

2 to which bit are you refering? – AJ. Oct 9, 2008 at 13:34

---

3 This is some cool stuff, but I wish you'd have some references for some claims. For example: I'd never heard the

optimize considers only 4 tables a time in a join. I don't understand how this could be right. Could you provide some references for that in particular? I'd love to see where you are getting this. – sheldonhull Jan 25, 2016 at 15:09

19

1. Have a pretty good idea of the optimal path of running the query in your head.
2. Check the query plan - always.
3. Turn on STATS, so that you can examine both IO and CPU performance. Focus on driving those numbers down, not necessarily the query time (as that can be influenced by other activity, cache, etc.).
4. Look for large numbers of rows coming into an operator, but small numbers coming out. Usually, an index would help by limiting the number of rows coming in (which saves disk reads).
5. Focus on the largest cost subtree first. Changing that subtree can often change the entire query plan.
6. Common problems I've seen are:
    - If there's a lot of joins, sometimes Sql Server will choose to expand the joins, and then apply WHERE clauses. You can usually fix this by moving the WHERE conditions into the JOIN clause, or a derived table with the conditions inlined. Views can cause the same problems.
    - Suboptimal joins (LOOP vs HASH vs MERGE). My rule of thumb is to use a LOOP join when the top row has very few rows compared to the

bottom, a MERGE when the sets are roughly equal and ordered, and a HASH for everything else. Adding a join hint will let you test your theory.

- Parameter sniffing. If you ran the stored proc with unrealistic values at first (say, for testing), then the cached query plan may be suboptimal for your production values. Running again WITH RECOMPILE should verify this. For some stored procs, especially those that deal with varying sized ranges (say, all dates between today and yesterday - which would entail an INDEX SEEK - or, all dates between last year and this year - which would be better off with an INDEX SCAN) you may have to run it WITH RECOMPILE every time.
- Bad indentation...Okay, so Sql Server doesn't have an issue with this - but I sure find it impossible to understand a query until I've fixed up the formatting.

Share Improve this answer Follow

answered Aug 20, 2008 at 21:48

community wiki
Mark Brackett

Slightly off topic but if you have control over these issues...

18

High level and High Impact.

- For high IO environments make sure your disks are for either RAID 10 or RAID 0+1 or some nested implementation of raid 1 and raid 0.
- Don't use drives less than 1500K.
- Make sure your disks are only used for your Database. IE no logging no OS.
- Turn off auto grow or similar feature. Let the database use all storage that is anticipated. Not necessarily what is currently being used.
- design your schema and indexes for the type queries.
- if it's a log type table (insert only) and must be in the DB don't index it.
- if your doing allot of reporting (complex selects with many joins) then you should look at creating a data warehouse with a star or snowflake schema.
- Don't be afraid of replicating data in exchange for performance!

Share Improve this answer Follow

answered Aug 20, 2008 at 21:14

community wiki
jason saldo

8

`CREATE INDEX`

Assure there are indexes available for your `WHERE` and `JOIN` clauses. This will speed data access greatly.

If your environment is a *data mart or warehouse,* indexes should abound for almost any conceivable query.

In a *transactional environment*, the number of indexes should be lower and their definitions more strategic so that index maintenance doesn't drag down resources. (Index maintenance is when the leaves of an index must be changed to reflect a change in the underlying table, as with `INSERT, UPDATE,` and `DELETE` operations.)

Also, be mindful of the order of fields in the index - the more selective (higher cardinality) a field, the earlier in the index it should appear. For example, say you're querying for used automobiles:

```
SELECT   i.make, i.model, i.price
FROM     dbo.inventory i
WHERE    i.color = 'red'
  AND    i.price BETWEEN 15000 AND 18000
```

Price generally has higher cardinality. There may be only a few dozen colors available, but quite possibly thousands of different asking prices.

Of these index choices, `idx01` provides the faster path to satisfy the query:

```
CREATE INDEX idx01 ON dbo.inventory (price, color)
CREATE INDEX idx02 ON dbo.inventory (color, price)
```

This is because fewer cars will satisfy the price point than the color choice, giving the query engine far less data to analyze.

I've been known to have two very similar indexes differing only in the field order to speed queries (firstname, lastname) in one and (lastname, firstname) in the other.

Share Improve this answer Follow

answered Aug 28, 2008 at 3:34

community wiki
Will SQL for Food

6

Assuming MySQL here, use EXPLAIN to find out what is going on with the query, make sure that the indexes are being used as efficiently as possible and try to eliminate file sorts. High Performance MySQL: Optimization, Backups, Replication, and More is a great book on this topic as is MySQL Performance Blog.

Share Improve this answer Follow

answered Aug 20, 2008 at 20:48

community wiki
davidmytton

3 That's good for MySQL, but the question was tagged "sqlserver". Still, it's a good thing to do that. The analogous thing to do in SSMS is to use "Display Estimated Execution Plan" and "Include Actual Execution Plan". If you can eliminate huge table scans and use clustered index seeks, then you're well on your way to optimal performance.
– eksortso May 15, 2009 at 22:48

6

A trick I recently learned is that SQL Server can update local variables as well as fields, in an update statement.

```
UPDATE table
SET @variable = column = @variable + otherColumn
```

Or the more readable version:

```
UPDATE table
SET
    @variable = @variable + otherColumn,
    column = @variable
```

I've used this to replace complicated cursors/joins when implementing recursive calculations, and also gained a lot in performance.

Here's details and example code that made fantastic improvements in performance: Link

Share Improve this answer Follow

edited Sep 9, 2022 at 14:06

community wiki
2 revs, 2 users 97%
jandersson

5

@Terrapin there are a few other differences between isnull and coalesce that are worth mentioning (besides ANSI compliance, which is a big one for me).

Coalesce vs. IsNull

Share Improve this answer Follow

answered Aug 20, 2008 at 23:33

community wiki
AlexCuse

4

Sometimes in SQL Server if you use an OR in a where clause it will really jack with performance. Instead of using the OR just do two selects and union them together. You get the same results at 1000x the speed.

Share Improve this answer Follow

answered Aug 20, 2008 at 20:56

community wiki
Ryan

I have seen this unexplained behavior. – Esen Mar 19, 2013 at 15:58

3

Look at the where clause - verify use of indexes / verify nothing silly is being done

```
where SomeComplicatedFunctionOf(table.Column) = @param
```

Share Improve this answer Follow

answered Aug 20, 2008 at 20:48

community wiki
Mike

I'll generally start with the joins - I'll knock each one of them out of the query one at a time and re-run the query

3

to get an idea if there's a particular join I'm having a problem with.

Share Improve this answer Follow

answered Aug 20, 2008 at 20:52

community wiki
John Christensen

3

On all of my temp tables, I like to add unique constraints (where appropriate) to make indexes, and primary keys (almost always).

```
declare @temp table(
    RowID int not null identity(1,1) primary key,
    SomeUniqueColumn varchar(25) not null,
    SomeNotUniqueColumn varchar(50) null,
    unique(SomeUniqueColumn)
)
```

Share Improve this answer Follow

answered Aug 20, 2008 at 21:00

community wiki
Seibar

2

@DavidM

> Assuming MySQL here, use EXPLAIN to find out what is going on with the query, make sure that

> the indexes are being used as efficiently as possible...

In SQL Server, execution plan gets you the same thing - it tells you what indexes are being hit, etc.

Share Improve this answer Follow

edited May 23, 2017 at 12:17

community wiki
2 revs
Seibar

2

Not necessarily a SQL performance trick per se but definately related:

A good idea would be to use memcached where possible as it would be much faster just fetching the precompiled data directly from memory rather than getting it from the database. There's also a flavour of MySQL that got memcached built in (third party).

Share Improve this answer Follow

answered Aug 20, 2008 at 20:56

community wiki
Andy

2

Make sure your index lengths are as small as possible. This allows the DB to read more keys at a time from the file system, thus speeding up your joins. I assume this works with all DB's, but I know it's a specific recommendation for MySQL.

Share Improve this answer Follow

answered Aug 20, 2008 at 21:01

community wiki
Barrett Conrad

2

I've made it a habit to always use bind variables. It's possible bind variables won't help if the RDBMS doesn't cache SQL statements. But if you don't use bind variables the RDBMS doesn't have a chance to reuse query execution plans and parsed SQL statements. The savings can be enormous: http://www.akadia.com/services/ora_bind_variables.html. I work mostly with Oracle, but Microsoft SQL Server works pretty much the same way.

In my experience, if you don't know whether or not you are using bind variables, you probably aren't. If your application language doesn't support them, find one that does. Sometimes you can fix query A by using bind variables for query B.

After that, I talk to our DBA to find out what's causing the RDBMS the most pain. Note that you shouldn't ask "Why is this query slow?" That's like asking your doctor to take out you appendix. Sure your query might be the problem, but it's just as likely that something else is going wrong. As developers, we we tend to think in terms of lines of code. If a line is slow, fix that line. But a RDBMS is a really complicated system and your slow query might be the symptom of a much larger problem.

Way too many SQL tuning tips are cargo cult idols. Most of the time the problem is unrelated or minimally related to the syntax you use, so it's normally best to use the cleanest syntax you can. Then you can start looking at ways to tune the database (not the query). Only tweak the syntax when that fails.

Like any performance tuning, always collect meaningful statistics. Don't use wallclock time unless it's the user experience you are tuning. Instead look at things like CPU time, rows fetched and blocks read off of disk. Too often people optimize for the wrong thing.

Share Improve this answer Follow

answered Aug 20, 2008 at 22:47

community wiki
Jon Ericson

2

First step: Look at the Query Execution Plan!
TableScan -> bad

NestedLoop -> meh warning
TableScan behind a NestedLoop -> DOOM!

SET STATISTICS IO ON
SET STATISTICS TIME ON

Share Improve this answer Follow

answered Sep 16, 2008 at 20:01

community wiki
Amy B

2

Running the query using WITH (NoLock) is pretty much standard operation in my place. Anyone caught running queries on the tens-of-gigabytes tables without it is taken out and shot.

Share Improve this answer Follow

answered Sep 22, 2008 at 14:26

community wiki
Valerion

2 This should be used judiciously, not habitually. Locking is not evil, just misunderstood. – user565869 Dec 12, 2014 at 23:29

2

Convert NOT IN queries to LEFT OUTER JOINS if possible. For example if you want to find all rows in Table1 that are unused by a foreign key in Table2 you could do this:

```
SELECT *
FROM Table1
WHERE Table1.ID NOT IN (
    SELECT Table1ID
    FROM Table2)
```

But you get much better performance with this:

```
SELECT Table1.*
FROM Table1
LEFT OUTER JOIN Table2 ON Table1.ID = Table2.Table1ID
WHERE Table2.ID is null
```

Share Improve this answer Follow

answered Jan 30, 2009 at 16:08

community wiki
Martin Brown

1

Index the table(s) by the clm(s) you filter by

Share Improve this answer Follow

answered Aug 20, 2008 at 20:52

community wiki

csmba

1

- Prefix all tables with dbo. to prevent recompilations.
- View query plans and hunt for table/index scans.
- In 2005, scour the management views for missing indexes.

Share Improve this answer Follow

answered Aug 20, 2008 at 20:58

community wiki
Stu

1

I like to use

```
isnull(SomeColThatMayBeNull, '')
```

Over

```
coalesce(SomeColThatMayBeNull, '')
```

When I don't need the multiple argument support that coalesce gives you.

http://blog.falafel.com/2006/04/05/SQLServerArcanaISNULLVsCOALESCE.aspx

Share Improve this answer Follow

answered Aug 20, 2008 at 21:03

community wiki
Seibar

1

I look out for:

- Unroll any CURSOR loops and convert into set based UPDATE / INSERT statements.
- Look out for any application code that:
  - Calls an SP that returns a large set of records,
  - Then in the application, goes through each record and calls an SP with parameters to update records.
  - Convert this into a SP that does all the work in one transaction.
- Any SP that does lots of string manipulation. It's evidence that the data is not structured correctly / normalised.
- Any SP's that re-invent the wheel.
- Any SP's that I can't understand what it's trying to do within a minute!

Share Improve this answer Follow

answered Aug 20, 2008 at 22:12

community wiki
Guy

1

```
SET NOCOUNT ON
```

Usually the first line inside my stored procedures, unless I actually need to use `@@ROWCOUNT`.

Share Improve this answer Follow

answered Aug 21, 2008 at 3:31

community wiki
travis

2 @@ROWCOUNT is set anyways. NOCOUNT disables the "xx rows affected" statements. – Sklivvz Oct 1, 2008 at 8:36

Does this really ever make an appreciable difference in performance? – JohnFx Jan 8, 2009 at 23:37

Yeah, then the count isn't automatically calculated every time a SQL statement is run. It's easy enough to bench a query with and without to see that it does make a difference. – travis Jan 16, 2009 at 20:44

The count is tracked in SQL Server anyway. Any performance difference that you see is because the counts have to go over the network to your front end. If you're doing a single SELECT it won't make an appreciable difference. If you have a loop with 100000 inserts it's a lot extra over the network. – Tom H Jan 21, 2009 at 15:14

1

In SQL Server, use the nolock directive. It allows the select command to complete without having to wait - usually other transactions to finish.

```
SELECT * FROM Orders (nolock) where UserName = 'momma'
```

Share Improve this answer Follow

answered Sep 19, 2008 at 16:01

community wiki
jinsungy

3 NOLOCK is only for queries for which you don't care about correct results – Mark Sowul Oct 19, 2012 at 13:15

1

Remove cursors wherever the are not neceesary.

Share Improve this answer Follow

edited Mar 5, 2009 at 17:40

community wiki
2 revs
Terrapin

Yeah, cursors are a curse! ;) – Sklivvz Oct 1, 2008 at 8:37

8 Ugh. Don't throw that out unqualified like that. Cursors are like guns. They aren't bad by themselves, its just that people do really bad things with them. – JohnFx Jan 8, 2009 at 23:36

1

Remove function calls in Sprocs where a lot of rows will call the function.

My colleague used function calls (getting lastlogindate from userid as example) to return very wide recordsets.

Tasked with optimisation, I replaced the function calls in the sproc with the function's code: I got many sprocs' running time down from > 20 seconds to < 1.

Share Improve this answer Follow

answered Aug 5, 2009 at 10:55

community wiki
callisto

0

Don't prefix Stored Procedure names with "sp_" because system procedures all start with "sp_", and SQL Server will have to search harder to find your procedure when it gets called.

Share Improve this answer Follow

edited Aug 20, 2008 at 21:16

community wiki
2 revs
Terrapin

1 Did you actually benchmark this one? If SQL Server is doing what is reasonable (using a hash algorithm to locate the Stored Proc), then this would make no difference. In fact if SQL Server *wasn't* doing that, it seems like system performance would stink (since it presumably calls it's own procs). – John Stauffer Sep 22, 2008 at 14:52

1 I think this falls in the bucket of premature optimization. It is probably a good practice to avoid confusion for people, but as an optimization tip... D- – JohnFx Jan 8, 2009 at 23:38

0

Dirty reads -

```
set transaction isolation level read uncommitted
```

Prevents dead locks where transactional integrity isn't absolutely necessary (which is usually true)

Share Improve this answer Follow

edited Jan 5, 2009 at 19:19

community wiki
2 revs
Terrapin

1 Yes, but this can lead to weird bugs which are VERY hard to find. – Grant Johnson Oct 27, 2008 at 18:12

0

I always go to SQL Profiler (if it's a stored procedure with a lot of nesting levels) or the query execution planner (if it's a few SQL statements with no nesting) first. 90% of the time you can find the problem immediately with one of these two tools.

Share Improve this answer Follow

answered Jan 30, 2009 at 15:42

community wiki
mwigdahl