

## Ten Top Tips for Speeding up SQL Server

My name's Jon Reade from SQL Server Club.

I'm here today because not every organisation has or needs a dedicated DBA.

So if you're a Developer or a Development DBA who doesn't have time for learning the in-depth complexities of SQL Server performance tuning, here's some tried and trusted techniques you can use in virtually every SQL Server environment.

### TIP 10 : Avoid using cursors and triggers

#### Why?

Because cursors and triggers operate on a row-at-a-time basis, they are inherently slow compared with set based TSQL operations. Whenever you see them, ask yourself "Is there a better way?"

#### How do I do it?

If you see a *cursor*, try to figure out a way to convert it to a set based TSQL operation.

If you see a *trigger*, look at how you can replace it by simpler, faster code wrapped up inside a stored procedure.

#### What's in it for me?

A big increase in the speed of your queries.

Your T-SQL skills improve, and you can squeeze more out of your current hardware.

You get to impress your difficult-to-impress DBA or manager.

#### What to watch out for? Some problems have to use cursors or triggers. Maybe 1%.

But if you think this is the case, ask a seasoned DBA if there's a better way.

**Cost:** Zero.

When you think cursor, think set based operation instead.

When you think trigger, think stored procedure.

## **TIP 9 : Use Backup Compression.**

### **Why?**

Because compressed backups not only save you disk space.

If you regularly backup and restore databases, they can buy you valuable test and development time over using native SQL Server backups, typically taking only 30-50% of the time that a native SQL Server backup or restore operation takes.

### **How?**

Upgrade to SQL Server 2008, or buy a third party tools such as SQL Backup / or SQL Litespeed.

Typical command :

Backup database Accounts to disk = 'F:\Accounts.bak' with compression

### **Obvious gain :**

Saves developer time backing up and restoring databases.

### **Not so obvious gain :**

As disk I/O is less, the disks get hit for less time, which means fewer user queries are affected whilst the backup is running.

### **Any gotchas?**

It's only available natively on the Enterprise and Developer Editions of SQL Server 2008. For other editions, buy third party solutions.

It uses more CPU and will often max out the CPU on single core boxes. So stick to using it on more powerful machines, unless your backup window permits, or you don't care that your server is unresponsive whilst backing up or restoring.

### **Cost?**

£135 - £200 per server for SQL Backup.  
Integrated into SQL Server 2008 at no extra cost.

**Read more about** Database backup compression

## **TIP 8 : Use Database Compression.**

### **New feature to SQL Server 2008.**

**Don't mix this up with backup compression !**

#### **Why use it?**

Because it reduces the size of your databases by compressing the data pages on-the-fly.

This means your server is moving fewer pages of data between disk and memory. THIS IS A GOOD THING !

#### **How do I do it?**

Upgrade to SQL Server 2008 to do this, as it's part of the internal database storage engine.

#### **What's in it for me?**

Smaller database files, and less disk I/O, which means faster access to data. Typically see 50-70% reduction in database size as a result.

#### **Gotchas?**

Uses more CPU and can max out single CPU boxes, so use it on more powerful machines.

It's only available on the Enterprise and Developer Editions of SQL Server 2008.

#### **Cost :**

Cost to upgrade to SQL Server 2008 - zero to thousands of pounds, depending on your edition, licensing or subscription deal.

#### **Tip :**

Try it out on SQL Server 2008 Developer Edition First - it's £45 on Amazon UK and is identical to the Enterprise Edition in every sense apart from the licensing. Read more : Database page compression

**Read more about** database compression, page compression

## **TIP 7 : Size your database!**

### **How?**

Set the database growth amount in the database properties dialog from Management Studio.

### **Why?**

As your database grows, it needs to be extended in size to accommodate the new rows. Each time a database grows, it hits query performance, and can cause the underlying database files to become fragmented in some situations.

### **How do I fix this problem?**

By sizing your database to be big enough from the start, you can reduce regular database extent growth.

This can also reduce disk fragmentation which is a primary but little considered cause of database performance problems.

Good values are typically 10Mb-100Mb, or 10%.

### **Watch out for...**

Very large databases, and databases with a high insert frequency.

Too little growth - database can become heavily fragmented, particularly if you've had to share the database server with an IIS server

Too much growth - easy to run out of disk space, large database extensions can cause queries to time out.

### **Cost?**

Zero.

**Read more about** Database auto-growth , database sizing

## **TIP 6 : Beware of SARGs!**

### **Why?**

Because how you write your Search ARGuments can drastically affect query time – with or without indexes.

### **How do I do it?**

#### **Attempt 1:**

```
select      Supplier
from        Accounts
where       Postcode like '%BS1%' -- XXX WRONG !!!
```

This is BAD because the % wildcard appears at the start of the where clause.

#### **Attempt 2:**

```
select      Supplier
from        Accounts
where       left(Postcode,3) = 'BS1' -- XXX ALSO WRONG !!!
```

This is also bad as the left hand side of the search arguments has to be evaluated, rather than expressing it as a column name. This can clobber search times.

#### **Attempt 3:**

```
select      Supplier
from        Accounts
where       Postcode like 'BS1%' -- MUCH BETTER :- )
```

### **Advantages?**

Searches will run MUCH quick if you keep a constant expression, ie : a value or column name on the left hand side of the equals. Likewise, keep % and \_ wildcards away from the left hand side of the search string on the right of the equals sign.

### **The Downside?**

Some where clauses may need to be re-written.

These are easy to spot – just search for " %" in your code if you embed your TSQL in your apps.

**Cost?** Zero.

**Read more about** SARGs, search arguments

## **TIP 5 : Keep Transactions Short**

If you need to use transactions, hold them open for the least time possible.

### **Why?**

Because held - open transactions can cause timeouts for other users, especially where lots of users hit the database simultaneously.

This problem is rarely picked up in testing.

### **How do I do it?**

Simple.

Store the data to be updated in data structures in the application code.

Only commit it to disk when the user's update is complete, in one or a series of small updates inside a single transaction block.

### **Upshot?**

Better application response times.

Fewer user complaints.

Less timeouts that look like they're a network problem (in reality they rarely are).

### **What should I watch out for?**

Don't open a database transaction, then wait for user input before closing it.

They might go off for lunch, which could cause a row or table lock to exist for an hour until they get back. This can lock out other users.

### **Cost?**

None – just change the way the code's written.

**Read more about** Pessimistic and optimistic locking

## **TIP 4 : Make Your Databases Read Only !**

Are you mad?

**No, not if the database is a copy of a live database restored just for reporting.**

### **Why?**

Because no row, page or table locking is used in read only databases, the overhead in running long queries is reduced substantially.

### **How do I do it?**

Restore the copy of your live database as usual.  
Right click on your database properties  
Set the read only attribute to True.

Or inside a scheduled job after restoring:

```
ALTER DATABASE AccountsReports SET READ_ONLY
```

### **Gain?**

Can be as much as 15-25% performance increase on those databases that are often hit the hardest.

### **WARNING!**

You can ONLY use it on databases that do not need to be written to.

### **Cost?**

Zero – just a minute of your time.

**Read more about** [locking, read only database performance](#)

### **TIP 3 : Develop against a copy of the live database.**

#### **Why?**

You get to see how quickly queries run against real data.

Test data is invariably a) Poor and b) small in size.

Upshot : You only find problems when you run the code against real data.

#### **How?**

Get your DBA to backup, copy and restore a copy of the live database to your dev / test boxes.

If this is too much for the network during the daytime, ask them to do it overnight on a scheduled job, or at the weekend.

#### **What's in it for me?**

You can often identify troublesome queries before they get into a live environment and you earn the wrath of your end users, along with an emergency fix.

**BUT !!** With VERY large databases, it can slow down the development process as queries can take a long time to run. However, it does show how the query will run in a live scenario, which is what ultimately matters.

#### **Gotchas**

Some DBAs will refuse to do this as data may be confidential or financially valuable, eg : medical or credit card information. In this case look to scrambling or encrypting identifiable or valuable data as part of the restore operation so that it cannot be easily exploited.

#### **Cost?**

Zero.

Some time might be involved removing sensitive data from the database before it can be given to the dev/test team.

## **TIP 2 : Ensure Join, Search and Order Columns Have Indexes**

If you're sorting on a specific column, searching for rows which contain a certain value in a column, or you're joining tables together on two or more columns, make sure that the columns have an index defined on them.

### **How do I do it?**

```
CREATE INDEX idxAccountNumber on tblAccount (intAccountNumber)
```

### **What's in it for me?**

Sort and join queries invariably work much faster if the columns being joined or sorted on are indexed. In large databases they are absolutely essential.

### **Do I have to do this?**

No, just replace your disks with Solid State Drives at 20x the cost per drive. But that's probably not an option in the current economic environment.

**WARNING!** Don't over index, it can slow down insert operations and can cause the indexes to grow to be larger than the data they're indexing.

### **The cost?**

Zero. Just think "Is there an index on this column" whenever you write a query which contains a join or order by statement. If there's not, you probably need one.

### **Bonus Tip :**

If you know a column will always contain unique values, use CREATE UNIQUE INDEX to get even better performance benefits.

**Read more about** indexes, clustered indexes, unique indexes, non clustered indexes, indexing strategies

## **TIP 1 : Get rid of dynamic SQL**

99% of TSQL code does not need to be generated dynamically.

### **How?**

Move your TSQL code out of your application and into a stored proc inside your database server. Then call the stored proc from your application and pass the parameters that would have been concatenated into the dynamic SQL.

### **The gain?**

Faster query response times, because SQL Server remembers the query plan it used last time the stored procedure was run, rather than having to figure it out again from scratch. This reduces CPU load on busy servers.

Improved SQL Server security because you can stop SQL Server users from executing ad-hoc TSQL.

Easier to troubleshoot a slow performing stored proc than it is dynamic TSQL.

### **Gotchas**

What should I watch out for? Occasionally dynamic SQL IS the only way. But not often. Just like cursors and triggers, always ask if it can be converted.

### **Cost?**

Zero. Although you'll need to dedicate some time to converting your TSQL into stored procs, and changing your application code to use the stored procs, it will be worth it.

**Read more about** [dynamic SQL](#), [execution plans](#), [plan re-use](#)

## Summary

- 10 Avoid Cursors and Triggers**
- 9 Use Backup Compression**
- 8 Use Database Compression**
- 7 Size your database !**
- 6 Beware of SARGS**
- 5 Keep Transactions Short**
- 4 Make Your Databases Read Only !**
- 3 Develop Against a Copy of Live**
- 2 Ensure Join, Search and Order Columns Have Indexes**
- 1 Get Rid of Dynamic SQL**

**TIP 0 : Rebuild your indexes out of hours, and only rebuild those that need updating !**

**How?**

Use DBCC DBREINDEX, DBCC INDEXDEFRAG, or use a tool such as Visual Defrag or SQL Defrag.

**What do I gain?**

Faster query response times.  
Fewer complaints from irate users.

**What to watch out for?**

Rebuilding indexes on a large database incurs a large disk I/O overhead, so rebuild them when the database server is not being used for end user queries or reporting.

**Cost?**

Zero if you do it yourself.  
£200-£1000 if you buy a tool to do it for you.

**Read more about** rebuilding indexes, DBCC commands

**OK, if you'd like to see the information in the slides, you can download them from Monday from the SQL Server Club site at the URL on the screen :**

**[www.SqlServerClub.com/ddd](http://www.SqlServerClub.com/ddd)**  
**[www.twitter.com/SQLServerClub](https://www.twitter.com/SQLServerClub)**

**My name's Jon Reade.**

**Thank you for your time, I hope you enjoy the rest of the day.**