50 ways to AVOID, FIND AND FIX ASP.NET PERFORMANCE ISSUES

ANTHONY DANG
KEVIN GRIFFIN
DAVID HANEY
TROY HUNT
ANTHONY VAN DER HOORN
MICHAEL SORENS
MATTHEW K
DAVE WARD
NICK HARRISON
JEREMY JARRELL
IAN DUNKERLY
TED JARDINE
ANTHONY MOORER
JULIE BELLER
GREGORY WHATLEY
SHAWN BINNS
BRUCE NORTON
STEPHEN KEANE

Foreword

This book began as a publishing experiment: could we make the collective wisdom of the ASP.NET and SQL Server communities available as an eBook? We chose performance improvements as our topic, and started asking for tips in November 2012.

You're reading the results now.

We'd like to thank our panel of judges, the LIDNUG group, and most of all the ASP.NET community for taking part. Although we set a strict timeframe for gathering and editing the advice that came in, there's no reason to stop here. Each tip came from a fellow developer and we'd love to gather more, so if you think something is missing, let us know, or use #50ASPtips to tweet about it. With your help, we can make ASP.NET apps run faster than Usain Bolt with cheetahs for shoes.

Michaela Murray,
.NET Tools Division at Red Gate Software.
dotnetteam@red-gate.com

redgate

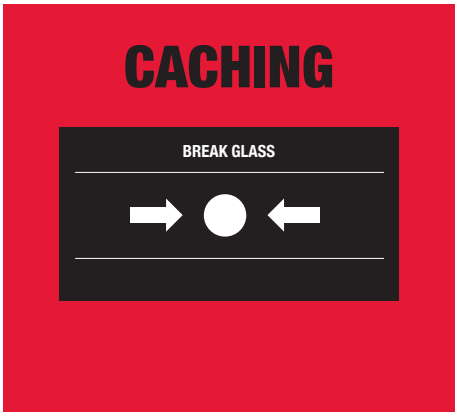
Contents

Caching is a last resort

Anthony Dang
@anthonydotnet

Projects that use multiple levels of cache often demonstrate a misunderstanding of why caching is required in the first place.

Caching is not synonymous with performance. Your code should already be efficient. Caching should only be used as a last resort, after you've made all possible (and sensible) code optimizations.



Remove unused View Engines

Kevin Griffin

@1kevgriff, www.kevgriffin.com

If you're an ASP.NET MVC developer, you might not know that ASP.NET still loads the View Engines for both Razor and Web Forms by default. This can cause performance issues because MVC will normally look for Web Forms views first, before switching over to the Razor views if it can't find them.

You can quickly eliminate this performance issue by adding the following two lines to your Global.asax, in the Application_Start():

```
ViewEngines.Engines.Clear();
ViewEngines.Engines.Add(
    newRazorViewEngine());
```

Goodbye Web Forms View Engine!

3

Use Microsoft's PDBs to debug or profile external assemblies or libraries

David Haney

www.haneycodes.net, www.linkedin.com/in/davidahaney

To accurately debug or profile an external assembly or library (i.e. one you're not directly compiling), you need the PDB files that accompany each of the DLLs. These files give your debugger or profiler access to information such as function names, line numbers, and other related metadata.

One thing that sucks in particular is debugging and profiling native Microsoft .NET assemblies without this kind of information. Fortunately, there's a solution for this very issue. With a little-known feature in Visual Studio 2012 (and 2010 too!), you can connect to Microsoft's Symbol Servers and obtain most of the debugging symbols for their assemblies and libraries.

Just go to Tools –> Options –> (expand) Debugging –> Symbols, and select the Microsoft Symbol Servers as your source for Symbols.

Getting the symbols from Microsoft every time you debug or profile is slow and painful. It'll even give you a pop-up saying as much once you check the Microsoft Symbol Servers, so be sure to specify a directory under "Cache symbols in this directory".

It will keep a local copy of the PDBs and check for updates every so often. As a result, you get your regular debugging/profiling AND you can see the function names of the Microsoft assemblies.

A selection of tips

Troy Hunt
Microsoft MVP

Make sure HTTP compression is turned on for any uncompressed content. HTML in particular compresses significantly, and in this era of mobile friendliness and slow 3G connections, that's essential.

Always set the CacheControlMaxAge attribute in web.config to a high number (a year is good). You don't want people pulling down the same static content they did last week. It'll also save on the bandwidth you're paying for.

Make use of the OutputCache annotation on MVC controllers. If the server can serve from memory, rather than going to disk or database, that's a good win.

Always profile your ORM database hits with SQL Profiler during development. ORMs get away from you very quickly. Before you know it, you've run a query 2000 times in a loop, when you could have retrieved all your data with a single database hit.

Watch out for lazy loading in ORMs. You shouldn't lazy load any entities that could be retrieved with a single database hit.

Implement different database queries in different contexts. In the API and on a webpage, you'll inevitably require different entity properties, so don't load things you don't need just because it's convenient to reuse a query.

Get MiniProfiler and configure it to always run when you hit your site (just don't enable it for the general public). You'll get detailed execution times and a big red warning if the same database query is running multiple times.

11

Make sure paging is conducted at the database layer

Anthony van der Hoorn

@anthony_vdh

When using grid UI controls (framework based, or 3rd party owned), you should carefully consider how paging is implemented. Many controls implement paging in a simplistic fashion, where the database is required to return all available data and the control limits what is shown. This strategy is fraught with performance problems, as it means that all the data in the given set must be extracted from the database (e.g. all customers or orders). Depending on the records involved, this could cause significant problems.

For a snappy user experience, always validate on the client

Simon Elliston Ball
@sireb

To avoid unnecessary round trips to the server, validate form entries on the client using JavaScript before posting them. This provides quick feedback and makes your application feel more responsive. Always make sure you explain your validation errors as well. If you use complex password rules or regex patterns, include a message to explain what the rules are to prevent user frustration.

13

Always perform validation on the server as well

Anthony van der Hoorn
@anthony_vdh

This isn't exactly a performance tip but rather a security tip for when people think that they could improve performance by cutting out server-side validation. These days, client-side validation can be bypassed with ease, so you can't trust what comes from the browser. So if you think you can save some processing cycles and bypass these steps, don't, as you're actually opening massive security holes.

Review what client scripts you are using

Anthony van der Hoorn

@anthony_vdh

Out of the box, many ASP.NET projects include client script libraries which you may or may not be using. It's always a good idea to check what you are using, and when.

Reduce memory leaks dramatically with the "using" statement

Michael Sorens

www.linkedin.com/in/michaelsorens

If a type implements IDisposable, wrap the use of it in a "using" statement, so that it automatically disposes of objects properly when the block exits.

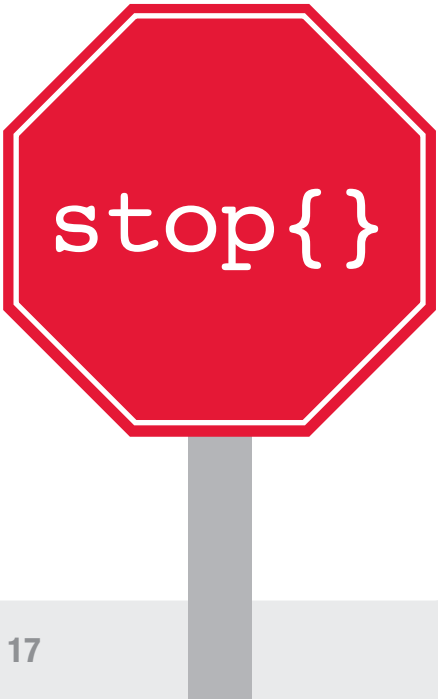
16

Reduce the data sent across the network

Matthew K

@mudnug

Reducing the amount of data sent across the network can improve application performance significantly. Compressing CSS and JavaScript is possible using bundling and minification. This will reduce the number of server requests and the amount of code sent across the wire.



17

Avoid running sites in debug mode

Dave Ward
@Encosia

When it comes to ASP.NET, one of the most common performance blunders I see on a regular basis is accidentally or intentionally running sites in debug mode. Language-level optimizations, such as using StringBuilders, Arrays instead of Lists, Switch instead of If-Then-Else, and so on, are popular, but when you measure their real impact, they usually pale in comparison to optimizations at the framework level.

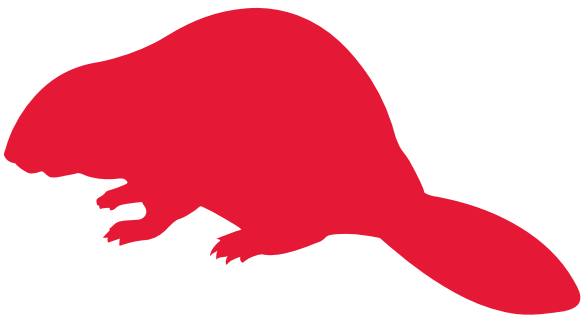
18

When in production, carefully consider what you need to log

Anthony van der Hoorn

@anthony_vdh

Many people deploy to production without checking how logging is currently configured. It is always advisable to check whether your policy is to have logging on or off by default and, if on, what level you should be targeting. In addition, you should check which targets you are outputting to, what archiving strategy you have, and whether your logging infrastructure allows for async logging.



A selection of tips

Nick Harrison
@neh123us

19

Including height and width in <img /> tags will allow your page to render more quickly, because space can be allocated for the image before it is downloaded.

20

Add script references at the bottom of the page, because asynchronous downloads halt when a script reference is reached. Style sheets and images can be downloaded asynchronously.

21

Use a content delivery network (CDN) for hosting images and scripts. They may be cached and it will reduce load on your server.

Use image sprites to retrieve smaller images in one download.

Use AJAX to retrieve components asynchronously that may not be needed immediately, such as the content of a collapsed panel, content behind a tab, and so on.

Make sure you've removed HTTP modules that aren't being used (Windows authentication, for example), and that you've disabled services such as FTP and SMTP, if you're not using them.

Use the startMode attribute to reduce the load time for your ASP.NET site

Jeremy Jarrell
@jeremyjarrell

Every time you update your site, IIS must recompile it during the first request, so the initial request takes significantly longer than subsequent ones. An easy solution is to tell IIS to automatically recompile your site as part of the update process. This can be done using the startMode attribute in the ApplicationHost.config file. You can even specify a custom action to run on start-up, such as pre-populating a data cache.

26

Don’t underestimate the value of the UI when tackling performance problems

Jeremy Jarrell
@jeremyjarrell

Simple UI tricks, such as progress bars, redirecting users’ attention using animation, or placing slower loading sections at the bottom of a page or off-screen, can often ‘fix’ a performance problem without the need to tune the underlying code. These UI tricks are an important tool to have in your performance tuning toolbox and can be much quicker and easier than addressing the underlying issue. They can act as a holdover until you have the time to devote to the core problem.

27

Throw hardware at the problem, not developers

Jeremy Jarrell
@jeremyjarrell

As developers, we often want to fix problems with code, but don't be afraid to 'put the compiler down' and throw some hardware at the problem. Performance problems caused by disk I/O bottlenecks or paging out of RAM can often be solved by a faster hard drive or more RAM. CPU-bound bottlenecks can often be solved by a new machine with a faster processor. As counter-intuitive as it sounds, addressing problems by buying a new machine or upgrading an aging one is often much cheaper than having a developer troubleshoot, diagnose, and correct a deep performance problem. And the rest of your website will get a performance kick to boot!

28

Don’t assume that problems can only arise from business logic

Jeremy Jarrell
@jeremyjarrell

When beginning to diagnose performance problems, we often assume the problem is in our business logic. Don’t forget that the areas of our code that provide infrastructure can cause problems as well. Areas such as HttpHandlers, HtmlHelpers, mapping, logging, or IoC frameworks are increasingly at the root of performance problems. While business logic still causes its share of problems, infrastructure code is quickly gaining in the performance problem race.

29

Before tackling any website performance issue, first verify the problem isn't on the client

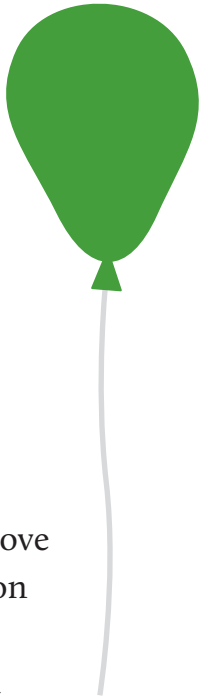
Jeremy Jarrell
@jeremyjarrell

Traditionally, many performance problems have been rooted in either the database or application server. However, with the proliferation of advanced JavaScript frameworks such as Backbone.js or jQuery, performance problems are increasingly starting to appear on the client. Rather than immediately attempting to diagnose a performance problem on the server, first use a free browser-based tool such as Google Chrome Developer Tools to ensure that the problem isn't actually occurring on the client. You may just save yourself a lot of time tracking down performance problems on the wrong end of your site.

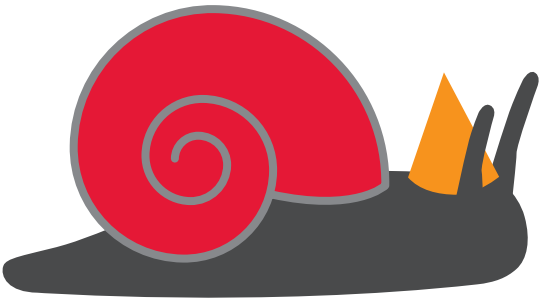
30

Static collections

Ian Dunkerly
@IDDesignsX



If a collection is static, make sure it only contains the objects you need. If the collection is iterated over often, then the performance can be slow if you don't remove unnecessary objects. Objects in a collection will be held in memory, even if they have been disposed of, which can also lead to a memory leak.



Know your loops

Ian Dunkerly
@IDDesignsX

`for` is the fastest way of iterating over a collection, `foreach` is a little slower, and LINQ queries are slowest.

Seven handy ViewState tips

Ted Jardine

@ovalsquare

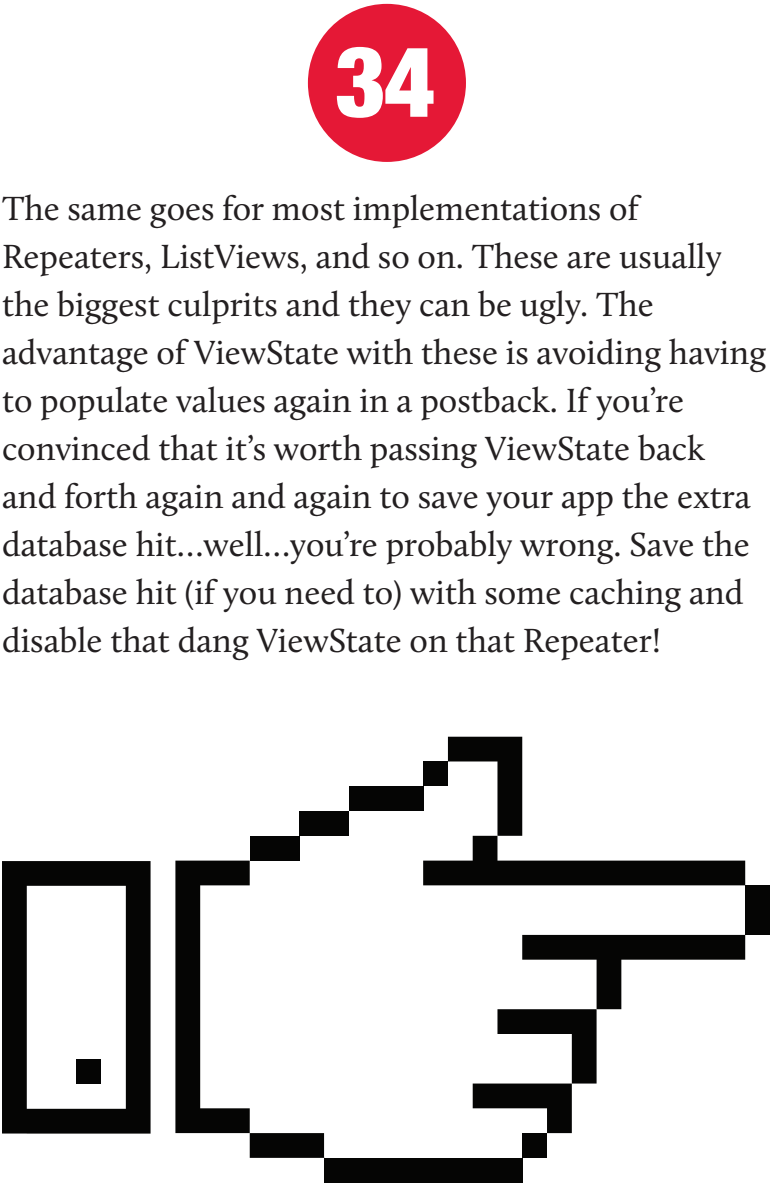
Every time I have to deal with a classic ASP.NET Web Forms application, one of the first things I look at is the resulting source, to check whether the DOM is a complete mess and whether the ViewState is an enormous, unnecessary blob of ugliness. Usually, they indicate what kind of mess will be found further down the stack.

```
<input type="hidden" name="__VIEWSTATE"
id="__VIEWSTATE" value="/wFzY3JpcH-
Q6IE9uY2xpY2s9J3dpbmRvdy5vcGVuKCJFcXVp-
cG1lbnQtRGV0YWlscy5hc3B4P1VzZWQtMjAxMC1UZ-
XJl…(continues for 18,000 characters)…
UlVTIiB3aWR0aD=" />
```

Yeah baby! ‘Nuff said

Inadvertently using ViewState when it's not necessary substantially increases the amount of data going back and forth, and can lead to a greater prevalence of invalid ViewState exceptions; the bigger the blob, the more likely it could be interrupted in transmission and not be posted back in entirety in a post.

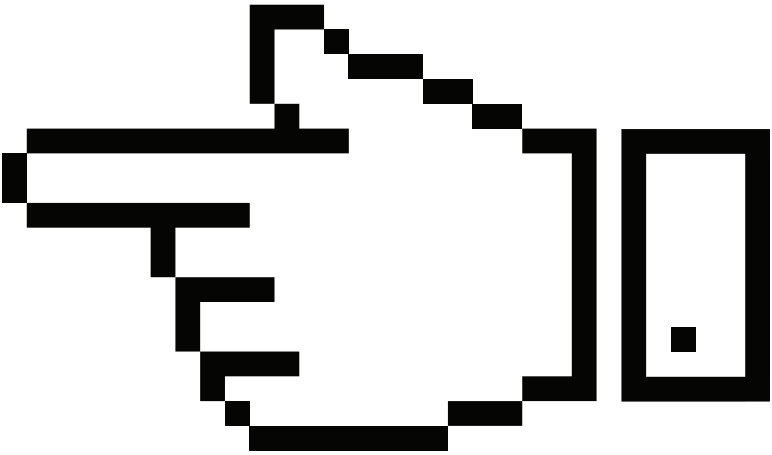
Unless you're tracking a Text_Changed event, you don't need ViewState enabled on TextBoxes and similar controls. Classic ASP.NET automatically repopulates TextBox.Text values upon postback, even without ViewState enabled. Turn it off on each TextBox with EnableViewState= "false" on each one. You can do this for other controls like labels, but unless you're setting their values after the page's load event, you won't reduce the size of the ViewState.



The same goes for most implementations of Repeaters, ListViews, and so on. These are usually the biggest culprits and they can be ugly. The advantage of ViewState with these is avoiding having to populate values again in a postback. If you're convinced that it's worth passing ViewState back and forth again and again to save your app the extra database hit…well…you're probably wrong. Save the database hit (if you need to) with some caching and disable that dang ViewState on that Repeater!

35

If you're re-binding data anyway, or just toggling one property on postback (asp:Panel anyone?), turn off that ViewState! Please!



If you do need ViewState, understand the page lifecycle and bind your data appropriately. A control loads its ViewState after Page_Init and before Page_Load, i.e. server controls don't start tracking changes to their ViewState until the end of the initialization stage. Any changes to ViewState mean a bigger ViewState, because you have the before value and the after value. So, if you're changing or setting a control's value, set it before ViewState is being tracked, if possible.

You may think it's impossible to turn off ViewState on a DropDownList, even if you re-bind it on every postback. But with a tiny bit of elbow grease you can keep ViewState enabled and avoid passing all your option values back and forth. This is particularly worthwhile for DropDownLists with a big ListItem collection. One way is to turn off ViewState and bind the select value manually to the actual posted value, like so:

```
string selectedId = Request[Countries.
UniqueID];
if (selectedId != null)
Countries.SelectedValue =
selectedId;
```

However, you may prefer something I came across more recently. Instead of binding your DropDownList in the typical Page_Load or Page_Init, bind it in the control's Init event:

```
<asp:DropDownList ID="Countries" ...
OnInit="CountryListInit" />
protected void CountryListInit(object
sender, EventArgs e)
{
Countries.DataSource = // get data
from database
Countries.DataBind();
}
```

Make it your habit to turn off ViewState on every control by default, and only turn it on when you need it. If a page doesn't need ViewState anywhere, turn it off at the page level. You do all that work to reduce requests, combine and compress static references, and make sure your code is as clean as possible - don't ruin it with a ViewState monster!

If you're anal, you can completely remove all traces of ViewState from pages that don't need it by inheriting from a BasePage such as this:

```
        /// <summary>
/// BasePage providing cross-site functionality
for pages that should not have ViewState enabled.
/// </summary>
public class BasePageNoViewState : Page // Or of
course, inherit from your standard BasePage, which in
turn inherits from Page
{
protected override void SavePageStateToPersistence
Medium(object viewState)
{
}
protected override object
LoadPageStateFromPersistenceMedium()
{
return null;
}
protected override void OnPreInit(EventArgs e)
{
// Ensure that ViewState is turned off for
every page inheriting this BasePage
base.EnableViewState = false;
base.OnPreInit(e);
}
}
    }
}
```

Avoid using session state

Anthony van der Hoorn
@anthony_vdh

Where possible, you should try and avoid using session state. Whilst using one web server, performance is usually not a problem. This changes as soon as you need to scale to multiple servers, as different, and usually slower, techniques need to be used.

Take advantage of .NET 4.5 async constructs

Anthony van der Hoorn

@anthony_vdh

With the arrival of .NET 4.5, writing async code correctly is easier than ever. Like any tool, it should be only applied where it makes most sense – in web use-cases this usually revolves around I/O operations (i.e. reading from disk, any network operation, database operations, or calls to web services).

41

StringBuilder is NOT the answer for all string concatenation scenarios; String.Join could be

Ted Jardine

@ovalsquare

Yes, if you are in a loop and adding to a string, then a StringBuilder *could* be most appropriate. However, the overhead of spinning up a StringBuilder instance makes the following pretty dumb:

```
var sb = new StringBuilder();
sb.Append("Frankly, this is ");
sb.Append(notMoreEfficient);
sb.Append(". Even if you are in a loop.");
var whyNotJustConcat = sb.ToString();
```

Instead, use String.Join, which is typically more performant than spinning up a StringBuilder instance for a limited number of strings. It's my go-to concat option:

```
string key = String.Join(“ “, new String[]
{ “This”, “is”, “a”, “much”, “better”,
solution, “.”});
```

The first variable of " " can just be set to "" when you don't want a delimiter.

For loops that do a lot of, er, looping, sure, use a StringBuilder. Just don't assume it's the de facto solution in all, or even the majority of cases. My rule of thumb is to add strings together when I've got one to five of them (likewise with String.Format if it helps with legibility). For most other cases, I tend towards String.Join. Only when dealing with a loop that isn't limited to about 10 iterations, especially one that really lets rip, do I spin up a StringBuilder.

ORM Tips

Anthony Moorer, Julie Beller, and Gregory Whatley

More and more people are using Object to Relational Mapping (ORM) tools to jump the divide between application code that is object oriented and a database that is storing information in a relational manner. These tools are excellent and radically improve development speed. But, there are a few 'gotchas' to know about.

Avoid following the 'Hello World' examples provided with your ORM tool that turns it into an Object to Object Mapping. Database storage is not the same as objects for a reason. You should still have a relational storage design within a relational storage engine such as SQL Server.

Parameterized queries are exactly the same as stored procedures in terms of performance and memory management. Since most ORM tools can use either stored procedures or parameterized queries, be sure you're coding to these constructs and not hard-coding values into your T-SQL queries.

Create, Read, Update, and Delete (CRUD) queries can all be generated from the ORM tool without the need for intervention. But, the Read queries generated are frequently very inefficient. Consider writing a stored procedure for complex Read queries.

Since the code generated from the ORM can frequently be ad hoc, ensure that the SQL Server instance has 'Optimize for Ad Hoc' enabled. This will store a plan stub in memory the first time a query is passed, rather than storing a full plan. This can help with memory management.

Be sure your code is generating a parameter size equivalent to the data type defined within table in the database. Some ORM tools size the parameter to the size of the value passed. This can lead to serious performance problems.

Database Performance Tips for Developers

As a developer you may or may not need to go into the database and write queries or design tables and indexes, or help determine configuration of your SQL Server systems. But if you do, these tips should help to make that a more pain free process.

T-SQL Tips

Shawn Binns and Bruce Norton

While much of your code may be generated, at least some of it will have to be written by hand. If you are writing some, any, or all of your T-SQL code manually, these tips will help you avoid problems.

SELECT * is not necessarily a bad thing, but it's a good idea to only move the data you really need to move and only when you really need it, in order to avoid network, disk, and memory contention on your server.

For small sets of data that are infrequently updated such as lookup values, build a method of caching them in memory on your application server rather than constantly querying them in the database.

Ensure your variables and parameters are the same data types as the columns. An implicit or explicit conversion can lead to table scans and slow performance.

Index Tips

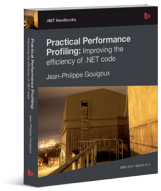
Stephen Keane and Gregory Whatley

Indexing tables is not an exact science. It requires some trial and error combined with lots of testing to get things just right. Even then, the performance metrics will change over time as you add more and more data.

You get exactly one clustered index on a table. Ensure you have it in the right place. First choice is the most frequently accessed column, which may or may not be the primary key. Second choice is a column that structures the storage in a way that helps performance. This is a must for partitioning data.

Performance is enhanced when indexes are placed on columns used in WHERE, JOIN, ORDER BY, GROUP, and TOP. Always test to ensure that the index does help performance.

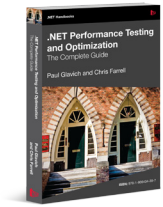
More free eBooks from Red Gate



Practical Performance Profiling: Improving the efficiency of .NET code

by Jean-Philippe Gouigoux

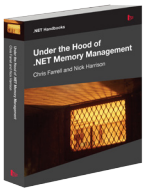
Theory and practical skills to analyze and improve the performance of .NET code. He guides the reader through using a profiler and explains how to identify and correct performance bottlenecks.



.NET Performance Testing and Optimization

by Paul Glavich and Chris Farrell

A comprehensive and essential handbook for anybody who wants to set up a .NET testing environment and get the best results out of it, or learn effective techniques for testing and optimizing.NET applications.



Under the Hood of .NET Memory Management

by Chris Farrell and Nick Harrison

Chris Farrell and Nick Harrison take you from the very basics of memory management, all the way to how the OS handles its resources, to help you write the best code you can.

Tools from Red Gate

ANTS Performance Profiler

Identify bottlenecks and optimize the performance of your application.

ANTS Memory Profiler

Find memory leaks and optimize the memory usage of your application.

.NET Reflector

Browse, analyse, decompile and debug your .NET code.

SmartAssembly

.NET obfuscator to protect your IP; plus, Error Reporting functionality to help you ship stable software by getting early user feedback.

.NET Demon

.NET Demon compiles your code continuously, so you see errors as soon as they are introduced.