What are the pros and cons to keeping SQL in Stored Procs versus Code [closed]

Asked 16 years, 4 months ago Modified 10 years, 10 months ago

Viewed 81k times

274

votes

Closed. This question is opinion-based. It is not currently accepting answers.

Closed 10 years ago.

Locked. This question and its answers are locked because the question is off-topic but has historical significance. It is not currently accepting new answers or interactions.

What are the advantages/disadvantages of keeping SQL in your C# source code or in Stored Procs? I've been discussing this with a friend on an open source project that we're working on (C# ASP.NET Forum). At the moment, most of the database access is done by building the SQL inline in C# and calling to the SQL Server DB. So I'm trying to establish which, for this particular project, would be best.

So far I have:

Advantages for in Code:

- Easier to maintain - don't need to run a SQL script to update queries
- Easier to port to another DB - no procs to port

Advantages for Stored Procs:

- Performance
- Security

c# sql sql-server stored-procedures

Share

edited Oct 27, 2008 at 14:46

community wiki
3 revs, 3 users 100%
Guy

50 You could also argue that Stored Procs make maintenance easier - you don't need to re-deploy the whole application just to change one query. – Darren Gosbell Oct 31, 2008 at 3:32

27 @GvS: that is a dysfunction of your company, not a best practice. Of course it is easier to change things in 1 place than 1000. The DBAs are simply doing their part to prevent cavalier changes to the system, and that should be respected. – user610217 Sep 7, 2011 at 14:27

47 Answers

Sorted by: Highest score (default)

1 2 Next

179 votes ✓

I am not a fan of stored procedures

> Stored Procedures are MORE maintainable because: * You don't have to recompile your C# app whenever you want to change some SQL

You'll end up recompiling it anyway when datatypes change, or you want to return an extra column, or whatever. The number of times you can 'transparently' change the SQL out from underneath your app is pretty small on the whole

> - You end up reusing SQL code.

Programming languages, C# included, have this amazing thing, called a function. It means you can invoke the same block of code from multiple places! Amazing! You can then put the re-usable SQL code inside one of these, or if you want to get really high tech, you can use a library which does it for you. I believe they're called Object Relational Mappers, and are pretty common these days.

> Code repetition is the worst thing you can do when you're trying to build a maintainable application!

Agreed, which is why storedprocs are a bad thing. It's much easier to refactor and decompose (break into smaller parts) code into functions than SQL into... blocks of SQL?

> You have 4 webservers and a bunch of windows apps which use the same SQL code Now you realized there is a small problem with the SQl code so do you rather...... change the proc in 1 place or push the code to all the webservers, reinstall all the desktop apps(clickonce might help) on all the windows boxes

Why are your windows apps connecting directly to a central database? That seems like a HUGE security hole right there, and bottleneck as it rules out server-side caching. Shouldn't they be connecting via a web service or similar to your web servers?

So, push 1 new sproc, or 4 new webservers?

In this case it *is* easier to push one new sproc, but in my experience, 95% of 'pushed changes' affect the code and not the database. If you're pushing 20 things to the webservers that month, and 1 to the database, you hardly lose much if you instead push 21 things to the webservers, and zero to the database.

> More easily code reviewed.

Can you explain how? I don't get this. Particularly seeing as the sprocs probably aren't in source control, and therefore can't be accessed via web-based SCM browsers and so on.

More cons:

Storedprocs live in the database, which appears to the outside world as a black box. Simple things like wanting to put them in source control becomes a nightmare.

There's also the issue of sheer effort. It might make sense to break everything down into a million tiers if you're trying to justify to your CEO why it just cost them 7 million dollars to build some forums, but otherwise creating a storedproc for every little thing is just extra donkeywork for no benefit.

Share

edited Sep 19, 2013 at 15:28

community wiki
4 revs, 2 users 98%
Orion Edwards

99
votes

This is being discussed on a few other threads here currently. I'm a consistent proponent of stored procedures, although some good arguments for Linq to Sql are being presented.

Embedding queries in your code couples you tightly to your data model. Stored procedures are a good form of contractual programming, meaning that a DBA has the freedom to alter the data model and the code in the procedure, so long as the contract represented by the stored procedure's inputs and outputs is maintained.

Tuning production databases can be extremely difficult when the queries are buried in the code and not in one central, easy to manage location.

[Edit] Here is another [current discussion]

Share

edited May 23, 2017 at 12:10

community wiki
2 revs
Eric Z Beard

47
votes

In my opinion you can't vote for yes or no on this question. It totally depends on the design of your application.

I totally vote against the use of SPs in an 3-tier environment, where you have an application server in front. In this kind of environment your application server is there to run your business logic. If you additionally use SPs you start distributing your implementation of business logic all over your system and it will become very unclear who is responsible for what. Eventually you will end up with an application server that will basically do nothing but the following:

```
(Pseudocode)

Function createOrder(Order yourOrder)
Begin
  Call SP_createOrder(yourOrder)
End
```

So in the end you have your middle tier running on this very cool 4 Server cluster each of them equipped with 16 CPUs and it will actually do nothing at all! What a waste!

If you have a fat gui client that directly connects to your DB or maybe even more applications it's a different story. In this situation SPs can serve as some sort of pseudo middle tier that decouples your application from the data model and offers a controllable access.

Share

edited Oct 28, 2010 at 7:56

community wiki
2 revs, 2 users 94%
huo73

44

votes

> Advantages for in Code:
>
> - Easier to maintain - don't need to run a SQL script to update queries
> - Easier to port to another DB - no procs to port

Actually, I think you have that backwards. IMHO, SQL in code is pain to maintain because:

- you end up repeating yourself in related code blocks
- SQL isn't supported as a language in many IDE's so you have just a series of un-error checked strings performing tasks for you
- changes in a data type, table name or constraint are far more prevalent than swapping out an entire databases for a new one
- your level of difficulty increases as your query grows in complexity
- and testing an inline query requires building the project

Think of Stored Procs as methods you call from the database object - they are much easier to reuse, there is only one place to edit and in the event that you do change DB providers, the changes happen in your Stored Procs and not in your code.

That said, the performance gains of stored procs is minimal as Stu said before me and you can't put a break point in a stored procedure (yet).

Share

edited Apr 17, 2009 at 14:10

community wiki
3 revs, 2 users 91%
Rob Allen

33
votes

CON

I find that doing lots of processing inside stored procedures would make your DB server a single point of inflexibility, when it comes to scaling your act.

However doing all that crunching in your program as opposed to the sql-server, *might* allow you to scale more if you have multiple servers that runs your code. Of-course this does not apply to stored procs that only does the normal fetch or update but to ones that perform more processing like looping over datasets.

PROS

1. Performance for what it may be worth (avoids query parsing by DB driver / plan recreation etc)
2. Data manipulation is not embedded in the C/C++/C# code which means I have less low level code to look through. SQL is less verbose and easier to look through when listed separately.

3. Due to the separation folks are able to find and reuse SQL code much easier.
4. Its easier to change things when schema changes - you just have to give the same output to the code and it will work just fine
5. Easier to port to a different database.
6. I can list individual permissions on my stored procedures and control access at that level too.
7. I can profile my data query/ persistence code separate from my data transformation code.
8. I can implement changeable conditions in my stored procedure and it would be easy to customize at a customer site.
9. It becomes easier to use some automated tools to convert my schema and statements together rather than when it is embedded inside my code where I would have to hunt them down.
10. Ensuring best practices for data access is easier when you have all your data access code inside a single file - I can check for queries that access the non performant table or that which uses a higher level of serialization or select *'s in the code etc.
11. It becomes easier to find schema changes / data manipulation logic changes when all of it is listed in one file.
12. It becomes easier to do search and replace edits on SQL when they are in the same place e.g. change /

add transaction isolation statements for all stored procs.

13. I and the DBA guy find that having a separate SQL file is easier / convenient when the DBA has to review my SQL stuff.

14. Lastly you don't have to worry about SQL injection attacks because some lazy member of your team did not use parametrized queries when using embedded sqls.

Share

edited Mar 2, 2010 at 19:24

community wiki
4 revs, 2 users 82%
computinglife

22
votes

The performance advantage for stored procedures is often negligable.

More advantages for stored procedures:

- Prevent reverse engineering (if created With Encryption, of course)
- Better centralization of database access
- Ability to change data model transparently (without having to deploy new clients); especially handy if multiple programs access the same data model

Share

answered Aug 18, 2008 at 19:57

community wiki
Stu

16
votes

I fall on the ***code*** side. We build data access layer that's used by all all the apps (both web and client), so it's DRY from that perspective. It simplifies the database deployment because we just have to make sure the table schema's are correct. It simplifies code maintenance because we don't have to look at source code and the database.

I don't have much problem with the tight coupling with the data model because I don't see where it's possible to really break that coupling. An application and its data are inherently coupled.

Share

answered Aug 18, 2008 at 20:29

community wiki
Rick

13
votes

Stored procedures.

If an error slips or the logic changes a bit, you do not have to recompile the project. Plus, it allows access from different sources, not just the one place you coded the query in your project.

I don't think it is harder to maintain stored procedures, you should not code them directly in the database but in separate files first, then you can just run them on whatever DB you need to set-up.

Share

answered Aug 18, 2008 at 20:01

community wiki
mbillard

24 If you find yourself making basic architectural decisions to avoid recompiling your code, then before doing anything at all, establish build process that doesn't totally suck. This is a non-argument. – Michael Borgwardt Apr 17, 2009 at 14:25

13 votes

Advantages for Stored procedures:

More easily code reviewed.

Less coupled, therefore more easily tested.

More easily tuned.

Performance is generally better, from the point of view of network traffic - if you have a cursor, or similar, then there aren't multiple trips to the database

You can protect access to the data more easily, remove direct access to the tables, enforce security through the procs - this also allows you to find relatively quickly any code that updates a table.

If there are other services involved (such as Reporting services), you may find it easier to store all of your logic in a stored procedure, rather than in code, and having to duplicate it

Disadvantages:

Harder to manage for the developers: version control of the scripts: does everyone have their own database, is the version control system integrated with the database and IDE?

Share

edited Feb 7, 2014 at 9:21

community wiki
2 revs, 2 users 94%
Matthew Farwell

Yes you can have a version control in Stored procedures and database in visual studio 2012 database project and tfs.
– hajirazin Jun 27, 2013 at 13:18

11
votes

In some circumstances, dynamically created sql in code can have better performance than a stored proc. If you have created a stored proc (let's say sp_customersearch) that gets extremely complicated with dozens of parameters because it must be very flexible, you can probably generate a much simpler sql statement in code at runtime.

One could argue that this simply moves some processing from SQL to the web server, but in general that would be a good thing.

The other great thing about this technique is that if you're looking in SQL profiler you can see the query you generated and debug it much easier than seeing a stored proc call with 20 parameters come in.

Share

answered Sep 5, 2008 at 16:01

community wiki
Joel Hendrickson

1 Not sure why this answer was voted down.. it's true that a smaller query can perform better. This has even been commented on by the SQL Server team. – Brannon Sep 25, 2008 at 7:53

9
votes

I like stored procs, dont know how many times I was able to make a change to an application using a stored procedure which didn't produce any downtime to the application.

Big fan of Transact SQL, tuning large queries have proven to be very useful for me. Haven't wrote any inline SQL in about 6 years!

Share

answered Aug 29, 2008 at 21:27

community wiki
Natron

I just can't understand another point of view using querries in code, just can't understand... – Chaki_Black Nov 26, 2013 at 21:13

8
votes

You list 2 pro-points for sprocs:

Performance - not really. In Sql 2000 or greater the query plan optimisations are pretty good, and cached. I'm sure that Oracle etc do similar things. I don't think there's a case for sprocs for performance any more.

Security? Why would sprocs be more secure? Unless you have a pretty unsecured database anyway all the access is going to be from your DBAs or via your application. Always parametrise all queries - never inline something from user input and you'll be fine.

That's best practice for performance anyway.

Linq is definitely the way I'd go on a new project right now. See this similar post.

Share

edited May 23, 2017 at 12:17

community wiki
2 revs
Keith

Ad-hoc SQL execution plans are only reused in specific circumstances: tinyurl.com/6x5lmd [SO answer with code proof] LINQ to SQL is officially dead: tinyurl.com/6298nd [blog post] – HTTP 410 Nov 9, 2008 at 11:34

1 Procs are by far the most secure way to go. They limit the user from only taking the actions in the procs. They can't go directly to a table or view that they have write access to and change things. Procs are necessary to prevent harm from internal threats. – HLGEM Aug 26, 2011 at 14:43

8
votes

@Keith

> Security? Why would sprocs be more secure?

As suggested by Komradekatz, you can disallow access to tables (for the username/password combo that connects to the DB) and allow SP access only. That way if someone gets the username and password to your database they can execute SP's but can't access the tables or any other part of the DB.

(Of course executing sprocs may give them all the data they need but that would depend on the sprocs that were available. Giving them access to the tables gives them access to everything.)

Share

answered Aug 18, 2008 at 21:12

community wiki
Guy

1 @Joe Philllips, views do not give you better or even equal security to procs and they will NOT be useful in preventing fraud or internal harm. When you use procs the scurity model is that they users only get access to the proc not the tables or views and thus they cannot do anything except what a proc does. If you have financial data and you aren't using procs, your system is at risk. – HLGEM Aug 26, 2011 at 14:41

7 votes

Think of it this way

You have 4 webservers and a bunch of windows apps which use the same SQL code Now you realized there is a small problem with the SQl code so do you rather...... change the proc in 1 place or push the code to all the webservers, reinstall all the desktop apps(clickonce might help) on all the windows boxes

I prefer stored procs

It is also easier to do performance testing against a proc, put it in query analyzer set statistics io/time on set showplan_text on and voila

no need to run profiler to see exactly what is being called

just my 2 cents

Share

answered Aug 18, 2008 at 20:16

community wiki
SQLMenace

6
votes

I prefer keeping in them in code (using an ORM, not inline or ad-hoc) so they're covered by source control without having to deal with saving out .sql files.

Also, stored procedures aren't inherently more secure. You can write a bad query with a sproc just as easily as inline. Parameterized inline queries can be just as secure as a sproc.

Share

answered Aug 18, 2008 at 20:01

community wiki
John Sheehan

6
votes

Use your app code as what it does best: handle logic.
User your database for what it does best: store data.

You can debug stored procedures but you will find easier to debug and maintaing logic in code. Usually you will end recompiling your code every time you change the database model.

Also stored procedures with optional search parameters are very inneficient because you have to specify in advance all the possible parameters and complex searches are sometimes not possible because you cant predict how many times a parameter is going to be repeated in the seach.

Share

edited Aug 18, 2008 at 22:26

community wiki
2 revs
Santi

---

2 When multiple apps hit the same database, and databases are affected by imports and other direct access (update all the prices by 10%), the logic must be in the database or you will lose database integrity. – HLGEM Sep 28, 2008 at 18:54

For the case of multiple apps, placing the logic into a library which is used by all apps allows integrity to be maintained while still keeping the logic in the app language. For imports/direct access, it's generally situational as to whether the rules which apply to apps should be enforced or not. – Dave Sherohman Oct 23, 2008 at 14:07

3 multiple apps shouldn't be making the same type of changes to the database. there should be one app component that deals with a single type of changes. Then that app should expose a service if others are interested. Multiple apps changing the same database/table in whatever way they see fit is what causes a system of apps and the database to become unmaintainable. – Jiho Han Feb 27, 2010 at 3:57

2 " there should be one app component that deals with a single type of changes" -- That component could be a stored

procedure, say in PL/SQL. – RussellH Oct 13, 2010 at 20:45

6
votes

When it comes to security, stored procedures are much more secure. Some have argued that all access will be through the application anyway. The thing that many people are forgetting is that most security breaches come from inside a company. Think about how many developers know the "hidden" user name and password for your application?

Also, as MatthieuF pointed out, performance can be much improved due to fewer round trips between the application (whether it's on a desktop or web server) and the database server.

In my experience the abstraction of the data model through stored procedures also vastly improves maintainability. As someone who has had to maintain many databases in the past, it's such a relief when confronted with a required model change to be able to simply change a stored procedure or two and have the change be completely transparent to ALL outside applications. Many times your application isn't the only one pointed at a database - there are other applications, reporting solutions, etc. so tracking down all of those affected points can be a hassle with open access to the tables.

I'll also put checks in the plus column for putting the SQL programming in the hands of those who specialize in it, and for SPs making it much easier to isolate and test/optimize code.

The one downside that I see is that many languages don't allow the passing of table parameters, so passing an unknown number data values can be annoying, and some languages still can't handle retrieving multiple resultsets from a single stored procedure (although the latter doesn't make SPs any worse than inline SQL in that respect).

Share

answered Sep 17, 2008 at 14:40

community wiki
Tom H

When the model changes, usually the code needs to change as well regardless of whether one is using sprocs or dynamic sql. And once you create tables/schema how often do you change just the table/schema? Not often. Usually changes come from business where they need to add another column or another table, in which case, I doubt you can do without a code change. – Jiho Han Feb 27, 2010 at 4:01

4
votes

One of the suggestions from a Microsoft TechEd sessions on security which I attended, to make all calls through stored procs and deny access directly to the tables. This approach was billed as providing additional security. I'm not sure if it's worth it just for security, but if you're already using stored procs, it couldn't hurt.

Share

answered Aug 18, 2008 at 20:10

community wiki
Eugene Katz

Data security is important when you are deling with personal information or finanical imformation especially. Most fraud is committed by insiders. You don't want to give them the access they need to bypass internal controls. – HLGEM Sep 28, 2008 at 19:00

4
votes

Definitely easier to maintain if you put it in a stored procedure. If there's difficult logic involved that will potentially change in the future it is definitely a good idea to put it in the database when you have multiple clients connecting. For example I'm working on an application right now that has an end user web interface and an administrative desktop application, both of which share a database (obviously) and I'm trying to keep as much logic on the database as possible. This is a perfect example of the DRY principle.

Share

answered Aug 18, 2008 at 20:15

community wiki
Andrew G. Johnson

4
votes

I'm firmly on the side of stored procs assuming you don't cheat and use dynamic SQL in the stored proc. First, using stored procs allows the dba to set permissions at the stored proc level and not the table level. This is critical not only to combating SQL injection attacts but towards preventing insiders from directly accessing the database and changing things. This is a way to help prevent fraud. No database that contains personal information (SSNs, Credit card numbers, etc) or that in anyway creates financial transactions should ever be accessed except through strored procedures. If you use any other method you are leaving your database wide open for individuals in the company to create fake financial transactions or steal data that can be used for identity theft.

Stored procs are also far easier to maintain and performance tune than SQL sent from the app. They also allow the dba a way to see what the impact of a database structural change will have on the way the data is accessed. I've never met a good dba who would allow dynamic access to the database.

Share

answered Sep 28, 2008 at 18:47

community wiki
HLGEM

4

votes

We use stored procedures with Oracle DB's where I work now. We also use Subversion. All the stored procedures are created as .pkb & .pks files and saved in Subversion. I've done in-line SQL before and it is a pain! I much prefer the way we do it here. Creating and testing new stored procedures is much easier than doing it in your code.

Theresa

Share

answered Dec 8, 2008 at 20:22

community wiki
Theresa

3

votes

Smaller logs

Another minor pro for stored procedures that has not been mentioned: when it comes to SQL traffic, sp-based data access generates *much* less traffic. This becomes important when you monitor traffic for analysis and profiling - the logs will be much smaller and readable.

Share

answered Sep 28, 2008 at 18:32

community wiki
Constantin

3
votes

I'm not a big fan of stored procedures, but I use them in one condition:

When the query is pretty huge, it's better to store it in the database as a stored procedure instead of sending it from the code. That way, instead of sending huge ammounts of string characters from the application server to the database, only the `"EXEC SPNAME"` command will be sent.

This is overkill when the database server and the web server are not on the same network (For example, internet communication). And even if that's not the case, too much stress means a lot of wasted bandwith.

But man, they're so terrible to manage. I avoid them as much as I can.

Share

answered Jun 8, 2010 at 9:15

community wiki
SiN

3
votes

A SQL stored proc doesn't increase the performance of the query

Share

answered Mar 24, 2011 at 10:43

community wiki

Gagnaire Eric

How it can be not ? Please Explain about this. – Sandesh Jan 27, 2014 at 12:24

It will increase the performance if the SQL query can be compiled. – TT. Feb 6, 2014 at 12:11

3

votes

Well obviously using stored procedures has several advantages over constructing SQL in code.

1. Your code implementation and SQL become independent of each other.
2. Code is easier to read.
3. Write once use many times.
4. Modify once
5. No need to give internal details to the programmer about the database. etc , etc.

Share

edited Oct 31, 2011 at 9:36

community wiki
2 revs, 2 users 71%
Bilal Khan

I've never been in a situation where having less info about a code problem or new feature has been of benefit to me, could you clarify number 5? – OpenCoderX Mar 21, 2013 at 1:32

2

votes

Stored Procedures are *MORE* maintainable because:

- You don't have to recompile your C# app whenever you want to change some SQL
- You end up reusing SQL code.

Code repetition is the *worst* thing you can do when you're trying to build a maintainable application!

What happens when you find a logic error that needs to be corrected in multiple places? You're more apt to forget to change that last spot where you copy & pasted your code.

In my opinion, the performance & security gains are an added plus. **You can still write insecure/inefficient SQL stored procedures.**

> Easier to port to another DB - no procs to port

It's not very hard to script out all your stored procedures for creation in another DB. In fact - it's *easier* than exporting your tables because there are no primary/foreign keys to worry about.

Share

answered Aug 18, 2008 at 20:44

community wiki
Seibar

Just a note: "Easier to port to another DB - no procs to port" referred to porting to *another DBMS*, not just another installation. There are alternatives out there, you know ;-).
– sleske Dec 8, 2009 at 10:47

2
votes

@Terrapin - sprocs are just as vulnerable to injection attacks. As I said:

> Always parametrise all queries - never inline something from user input and you'll be fine.

That goes for sprocs and dynamic Sql.

I'm not sure not recompiling your app is an advantage. I mean, you have run your unit tests against that code (both application and DB) before going live again anyway.

@Guy - yes you're right, sprocs do let you control application users so that they can only perform the sproc, not the underlying action.

My question would be: if all the access it through your app, using connections and users with limited rights to update/insert etc, does this extra level add security or extra administration?

My opinion is very much the latter. If they've compromised your application to the point where they can re-write it they have plenty of other attacks they can use.

Sql injections can still be performed against those sprocs if they dynamically inline code, so the golden rule still applies, all user input must always be parametrised.

Share

answered Aug 18, 2008 at 21:32

community wiki
Keith

It isn't just outside atacks you need to fight. You cannot allow direct access to the tables to users who could then alter the data to commit fraud. – HLGEM Sep 28, 2008 at 18:56

AS a policy, stored procs should not be allowed to use dynamic sql, there is almost always a non-dynamic solution if you look for it. – HLGEM Sep 28, 2008 at 18:57

SQL injection is not so effective against sprocs with dynamically-inlined code because dynamic code executes with caller permission, not owner permission (unlike static code). This is true for SQL Server - not sure about Oracle.
– HTTP 410 Nov 9, 2008 at 11:38

2 votes

Something that I haven't seen mentioned thus far: the people who know the database best aren't always the people that write the application code. Stored procedures give the database folks a way to interface with programmers that don't really want to learn that much about SQL. Large--and especially legacy--databases aren't the easiest things to completely understand, so programmers might just prefer a simple interface that gives them what they need: let the DBAs figure out how to join the 17 tables to make that happen.

That being said, the languages used to write stored procedures (PL/SQL being a notorious example) are pretty brutal. They typically don't offer any of the niceties you'd see in today's popular imperative, OOP, or functional languages. Think COBOL.

So, stick to stored procedures that merely abstract away the relational details rather than those that contain business logic.

Share

answered Sep 5, 2008 at 16:17

community wiki
yukondude

"The languages used to write stored procedures (PL/SQL being a notorious example) are pretty brutal [and] don't offer any of the niceties you'd see in today's popular languages." You need to re-read the PL/SQL docs (download.oracle.com/docs/cd/B28359_01/appdev.111/b28370/toc.htm). PL/SQL has encapsulation using packages, OOP via object types, exception handling, dynamic execution of code, debuggers, profilers, etc., plus hundreds of standard, Oracle-supplied packages/libraries for doing everything from HTTP calls to encryption and regular expressions. PL/SQL has LOTS of niceties. – ObiWanKenobi Aug 5, 2009 at 22:51

2
votes

I generally write OO code. I suspect that most of you probably do, too. In that context, it seems obvious to me that all of the business logic - including SQL queries - belongs in the class definitions. Splitting up the logic such that part of it resides in the object model and part is in the database is no better than putting business logic into the user interface.

Much has been said in earlier answers about the security benefits of stored procs. These fall into two broad categories:

1) Restricting direct access to the data. This definitely is important in some cases and, when you encounter one, then stored procs are pretty much your only option. In my experience, such cases are the exception rather than the rule, however.

2) SQL injection/parametrized queries. This objection is a red herring. Inline SQL - even dynamically-generated inline SQL - can be just as fully parametrized as any stored proc and it can be done just as easily in any modern language worth its salt. There is no advantage either way here. ("Lazy developers might not bother with using parameters" is not a valid objection. If you have developers on your team who prefer to just concatenate user data into their SQL instead of using parameters, you first try to educate them, then you fire them if that doesn't work, just like you would with developers who have any other bad, demonstrably detrimental habit.)

Share

answered Oct 23, 2008 at 14:30

community wiki
Dave Sherohman

2
votes

I am a huge supporter of code over SPROC's. The number one reasons is keeping the code tightly coupled, then a close second is the ease of source control without a lot of custom utilities to pull it in.

In our DAL if we have very complex SQL statements, we generally include them as resource files and update them as needed (this could be a separate assembly as well, and swapped out per db, etc...).

This keeps our code and our sql calls stored in the same version control, without "forgetting" to run some external applications for updating.

Share

edited Jul 7, 2010 at 14:56

community wiki
2 revs, 2 users 89%
Tom Anderson

What about when you "forget" to replicate changes to tables? – craigmoliver Aug 17, 2011 at 3:38

Process can solve deployment issues. – Tom Anderson Aug 17, 2011 at 21:52