SQL queries not in one place but assembled dynamically while passing through several system layers. Is this a good practice?

Asked 15 years, 9 months ago Modified 15 years, 9 months ago

Viewed 387 times

0

I personally find that makes the life of a developer who has recently joined the project a very sad one. Without almost perfect understanding of the framework mechanisms it makes developing and debugging like a torture, since whenever I get an error or an unexpected behavior I have not a slightest idea where to look. In some rare cases Ctrl+F through the solution helps, but in most cases I either have to ask the senior guy or use try & error approach. In many cases there is no way to test whether it's working save and nice, testers and sadly customers have to check it.

I think that putting queries in stored procedures or at least in one place in code could help.

Is this dynamic approach to queries a standard practice in business applications? I'm personally quite discomfortable with it.

asp.net sql

Share

Improve this question

Follow

asked Mar 16, 2009 at 14:46

User
30.9k ● 22 ● 81 ● 108

7 Answers

Sorted by: Highest score (default)

2

✓

If its a nightmare to debug, then you already have your answer: its not a best practice, and the pattern should be avoided in the future.

For what its worth, dynamic SQL is not necessarily a bad thing, and its extremely common in large business applications. When its done right, it can improve the maintainability of code:

- Most ORMs like Hibernate make database access transparent to the programmer. For example, when you request an object like `User.GetByID(12)`, the ORM will dynamically construct the SQL, execute it, map the fields to your object, and return it. As a programmer, a huge amount of your time is freed up because you don't need to write SQL queries anymore, you can just focus on writing your application. **You can query the database without seeing, touching, or smelling a hardcoded SQL string anywhere in your application, so that you maintain a hard separation between business**

logic and your data access layer without writing stored procedures.

- If your dynamically constructed SQL is well-abstracted, then you can swap out databases without changing your data access logic. NHibernate, for example, generates correct SQL based on whatever database vendor is listed in your config file, and it simply just works. **You can, in principle, use the same data access logic with any database vendor.**
- I've seen applications which contain 1000s of stored procedures representing trivial queries (i.e. simple selects, inserts, updates, and deletes). Maintaining these databases is beyond cumbersome, especially when your database schema changes frequently. With well-written dynmamic SQL, you can represent 1000s of equivalent queries in just a few functions (i.e. a function where you pass a tablename and some params). ORMs already provide this abstraction for you, so changes to your database schema do not require changes to your application code (unless you want to add that new field to your business object, of course). **You can change your database schema (add new fields, change datatypes of columns, etc) without changing your application code or data access logic.**
- There are some smaller benefits as well:
    - Regardless of whatever whatever policies there are which require programmers or CMs to copy

stored procedures into source control, we forget. If SQL is built up programmatically, its always in source control.

- There are lots of myths about the advantages of stored procedures over dynamic SQL:
    - "Dynamic SQL is prone to SQL injection, stored procedures are not" - simply not true. Dynamic SQL is trivial to parameterize, and in fact parameterized dynamic SQL is easier to write than dynamic SQL which uses excessive string concats. Even still, if you work with a programmer who is so negligent that he writes unsafe dynamic SQL, then chances are he'll write stored procedures just as badly (see this as a case study).
    - "Stored procedures are pre-compiled, so they run faster. SQL Server can't cache the execution plan of dynamic SQL" - at least with SQL Server, this is absolutely wrong. Since SQL Server 7.0, "a stored procedure is compiled at execution time, like any other Transact-SQL statement. SQL Server 2000 and SQL Server 7.0 retain execution plans for all SQL statements in the procedure cache, not just stored procedure execution plans."
    - "Stored procedures let you change SQL without redeploying an application." - this is true for very trivial tweaks, but in 95% of cases, changes to

the database require application changes anyway.

Dynamic SQL isn't bad *per se*, but there are lots of warning signs which indicate when you're doing it wrong:

- You should *never* see regex to dissect an SQL statement.
- You should *never* pass chunks of SQL to your data access layer. For example, a method with the signature `User GetUser(string whereclause)` is very fragile. If your database schema changes, your application logic changes as well.
- Data access should be typesafe. You should never pass around untyped datasets or strings -- or if you do, you need to wrap these objects in a typed object before returning them to the user.
- Always consider using an existing ORM before rolling your own.

You've probably noticed I've talked a lot about ORMs in this post. That's because, in general, hard-coding SQL strings is a *bad thing*. The correct way to use dynamic SQL is to have a tool which generates SQL on-the-fly, which is exactly what many ORMs can and already do.

Share Improve this answer Follow

edited May 23, 2017 at 12:19

Community Bot
1 ●1

answered Mar 16, 2009 at 16:14

Juliet
81.4k ● 46 ● 199 ● 229

2

One of the big problems I see with this approach is that there is no clear separation of responsibility. With multiple layers owning different parts of the logic that creates a SQL query you have an application that will not scale well and will be difficult to modify.

Share Improve this answer Follow

answered Mar 16, 2009 at 14:57
Andrew Hare
351k ● 75 ● 645 ● 641

1

Dynamic queries can also open you up to SQL injection. Having them in stored procedures with paramters helps reduce that risk.

If you do need/want to do queries dynamically in code, I'd suggest having them all within a single data class or similar. This way they are all in one place if any changes are needed, plus then you can see if an existing query already exists before making a new one. This also helsp if you ever need to use a new database or want to switch to store procedures, sicne everything is wrapped within calls in a single location.

Share Improve this answer Follow

answered Mar 16, 2009 at 14:48
schooner
3,077 ● 8 ● 32 ● 39

Dynamic SQL can be parameterized with trivial effort. Only the most ridiculously negligent programmers will make an application vulnerable to SQL injection using dynamic SQL -- truly awful programmers can (and often do) write stored procedures which leave apps open to SQL injection as well. – Juliet Mar 16, 2009 at 14:58

I agree, Iwas was just saying that in code dynamic sql can open you up to injection, SP often reduce this. An truly aweful programmer can make a mess of anything. – schooner Mar 16, 2009 at 15:00

1

You're not wrong, I personally think you should either use stored procedures or have a class specifically for talking to the database. Just assembling queries "wherever" in the code is asking for trouble.

Share Improve this answer Follow

answered Mar 16, 2009 at 14:50

Kevin Laity
2,481 ● 1 ● 27 ● 36

"Wherever" is just a part of the problem. They are assembled "everywhere" making my wonder where the pieces came from and why they were put there... – User Mar 16, 2009 at 14:54

Exactly, it sounds like there should have been a much more cohesive strategy here. – Kevin Laity Mar 16, 2009 at 15:01

In my projects I actually use both methods, I have a single class for calling the database, and it calls it using stored procedures only. – Kevin Laity Mar 16, 2009 at 15:02

0

Stored procedures are a must. Two principles at play here: good engineering and security

Databases are not OOP nor functional programming languages either (though they pretend to be). A database's strength is in it's ACID nature. To utilize this power, DBs really need their own layer to adhere to those principles.

Secondly, as others have mention, data injection attacks are a HUGE threat. When you have millions of dollars on the line, it is criminally negligent to your share holders or clients that such data isn't protected through use of stored procedures or some type of safe databinding.

If you HAVE to dynamically build a query, the best way to make it secure is to only allow input to be translated to a known integer/enum value. No free text input.

It is easy to normalize a bit of user input and check against known values in an array. I usually do this across multiple layers so that the input is checked multiple times and normalized multiple times.

But again. Stored procedures are a must.

Share Improve this answer Follow

answered Mar 16, 2009 at 16:28

thesmart

3,063 ● 2 ● 33 ● 34

-1: The reference to database's ACID nature makes no sense. What does it have to do with stored procedures?

Additionally, SQL can be parameterized in *exactly* the same way as stored procedures to prevent injection, and as an added bonus parameterizing dynamic SQL allows you to accept string inputs. – Juliet Mar 19, 2009 at 16:27

The reason I make reference to ACID is because each procedure should ahead to ACID -- specifically, each procedure should be atomic and operations should not span procedures should another layer (such as php) decide to bork. – thesmart Mar 19, 2009 at 21:26

-1

Although everyone seems to be pointing out SQL injection attacks as a big security vulnerability, that's not the biggest security hole caused by dynamic SQL in my opinion. You can prevent SQL injection attacks if you code the front end properly and without coding your stored procedures properly you can still leave yourself vulnerable.

I think that the biggest security hole in using dynamic SQL is that now the front end user needs to have access to the underlying tables. That means that there is a user out there, whether it's Windows Authentication or a SQL Login, that can muck with the database with almost no rules to stop them. I can't even count the number of times I've gone into a new shop and found that the login information was kept in a config file somewhere in plain text. Even when it wasn't, the developers almost always knew the username/password for the account.

Most security breaches (by a LONG shot) in corporations are inside jobs - disgruntled employees, people looking to make a quick buck by selling SSNs, or people who just don't think there's anything wrong with looking up medical information for their ex-girlfriend from high school. Add to that the developer who issued a "TRUNCATE TABLE" when he was **sure** that he was on the development server. Then you've also got the people who login and add a row to a table somewhere to "fix" a problem, but they don't understand that whenever you add a row to table X you need to write a row to table Y. Had all access been enforced through stored procedures they wouldn't have been able to do that.

Share Improve this answer Follow

answered Mar 16, 2009 at 19:03

Tom H

47.4k ● 15 ● 89 ● 131

-1: You appear to be criticizing dynamic SQL, but you're making a different argument entirely. Stored procedures do not prevent programmers from storing DB credentials in the config file, from mucking around with the database, or granting database/table permissions through windows authentication. – Juliet Mar 19, 2009 at 16:41

Try designing an application that DOESN'T have direct table permissions granted without using stored procedures. It's not a matter of SPs PREVENTING direct permissions, it's a matter of it being impossible to NOT have direct permissions without SPs. – Tom H Mar 19, 2009 at 19:00

-4

i would like to add that now we have hibernate and linq stored procedures are overkill in most projects. i understand in some situations stored procedures are unavoidable because of performance constraints since shipping masses of data back and forward between the application server and the database is expensive.

my current project uses SPs and my previous job used hibernate. on the current job there are FAR more bugs a lot of them that would never have occured if we were using an ORM. also, on the SP project we had a SQL injection bug which was in a stored procedure. i've never seen an SQL injection problem on a codebase i have worked on before now :) it seems that people's brains just turn off when they start writing stored procedures.

why would you want to have more bugs and do more work just to please some screwed up religious principle: 'we must use stored procedures'.

Share Improve this answer Follow

answered Mar 16, 2009 at 16:45

benmmurphy
2,505 ●1 ●21 ●31