Rated
Viewed 18124 times

Actions
Save as Favourite
Share with a Friend
PDF Version
Report Problem

Contents

To SP or not to SP in SQL Server - Introduction

Author Douglas Reilly

Introduction

This article is reprinted with permission from Simple Talk, the journal for Microsoft technology developers and DBAs.

A number of years ago, I went to work for SportSoft Golf (later Golf Society of the U.S.), one of the dot-com companies of old. Like many dot-com companies, SportSoft Golf failed not long after September 11th. While working there, the change from doing both fat client and intranet development to doing only Internet development was jarring, as was the change to using only stored procedures (SPs) and moving away from ad-hoc SQL.

Jim, my boss at the time, was a longtime SQL Server developer, and was most comfortable working inside SQL Server. I initially resisted but was eventually won over, and to this day virtually all of my database access code uses SPs.

The topic of using or not using SPs does not reach the level of religious fervor in many quarters. I became aware of the debate back in November 2003 when Rob Howard, then a Microsoft Program Manager and now the founder of Telligent systems, posted a lengthy blog entry on why all SQL Server developers should be using SPs. Frans Bouma, a Microsoft MVP and developer of LLBLGen Pro, an O/R mapper, posted his rebuttal. There have been a number of discussions since then, often shedding more heat than light.

The biases

Not to pick on Rob and Frans but rather to illustrate the biases of a number of the folks involved in the SP versus ad-hoc SQL debate, I will use these gentlemen as examples. I know Rob personally and like him. I have met Frans only in the virtual world, but must say I respect his abilities, and even when I disagree with him, I find him agreeable. That said, both advocates have some bias that I imagine influences the way they feel about SPs. Rob's is pretty obvious. He was a Microsoft employee when he wrote the blog entry mentioned above, and even now he does a fair amount of business with Microsoft. Microsoft has an interest in folks taking full advantage of all that SQL Server has to offer (including SPs), as a way to lock those people into using Microsoft SQL Server. Frans develops an O/R mapper, and doing an O/R mapper that needs to support multiple databases is clearly easier using ad-hoc SQL.

They are both pretty smart people, smarter than most.

The Security Argument

One of the most frustrating things about watching the SP or not debate is how misused this particular argument is for the use of SPs. For instance, Rob says:

> *What permissions are required for working with your data? If embedded SQL is used it likely means that the application can execute any `INSERT`, `UPDATE`, `DELETE`, `SELECT` script it desires. You wouldn't – hopefully you don't – run as the administrator (sa) account on your server, so why allow full access to the database itself?*

Some of what Rob says is true. Many people probably do give full access to all tables to the user who is accessing his or her database from his or her application. This is a terrible practice, and it is unlikely that the use of SPs or anything else will help. What Rob misses is that you can use the very fine-grained user-level security allowed by SQL Server to prevent people from doing a `SELECT` or `DELETE` against all tables. Frans mentions the use of Views as an alternative for allowing certain users access to only certain columns of a table.

Unfortunately, this does not always address the problem of allowing users access only to certain rows of a table. For instance, if you have an Employee table, you can grant access to `SELECT` from a view that shows demographic information, while not showing salary information. More difficult is restricting the view to only some rows of the Employee table. If you have four regions, it might be reasonable to have a `vwEastEmployees`, `vwCentralEmployees`, and so on. If, however, you have dozens of small regions, creating all those views and maintaining the security on them is not fun.

I have a number of situations in which users can have access to only certain rows of a database. The rules covering which rows a user can access are often complex and best managed in procedural code, or at least not easily handled in straight SQL Set-based code. In these situations, placing this logic in an SP (and denying any access to the bare tables) is very helpful.

One of the most damaging arguments raised in defense of SPs is that they somehow magically prevent SQL injection attacks. From Rob's post:

> *Additionally, stored procedures are a counter-measure to dangerous SQL Script injection attacks, a susceptibility that applications using embedded SQL are more vulnerable to.*

Sorry, but this is just not true. Using SPs make it more likely that you will pass parameters the right way, but there is no guarantee. For instance, this is some code I recently read answering a question on http://www.asp.net :

```
strsql = "EXECUTE findtitle '" & textboxtitle.text & "'"
objCmd = New SqlCommand(strSQL, objConn)
```

Trust me, even though this hapless programmer is using stored procedures, the application is susceptible to SQL injection attacks.

In the Microsoft SQL Server environment, SQL injection attacks can be prevented using parameters, with or without SPs. Earlier I said this is a damaging argument, and by that I mean it is damaging to programmers who cannot use SPs: They will leave their applications more vulnerable to attack than they should because of this bit of misinformation.

Security: No SP advantage, unless restricting access to rows in complex ways.

The Performance Argument

The next argument often raised by SP advocates is that SPs offer better performance. Rob says:

> *When stored procedures are used SQL Server can cache the 'execution plan' that it uses to execute the SQL vs. having to recalculate the execution plan on each request – would you recompile your business logic class on each request? …probably not.*

Note that Rob initially used the word "pre-compile or cache" when describing SPs, which he has edited out of the post, as noted by Rob at the bottom of the entry.

The counter to this is provided by Frans, who quotes SQL Server Books On-Line, as I do here:

> *SQL Server 2000 and SQL Server version 7.0 incorporate a number of changes to statement processing that extend many of the performance benefits of stored procedures to all SQL statements. SQL Server 2000 and SQL Server 7.0 do not save a partially compiled plan for stored procedures when they are created. 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. The database engine uses an efficient algorithm for comparing new Transact-SQL statements with the Transact-SQL statements of existing execution plans. If the database engine determines that a new Transact-SQL statement matches the Transact-SQL statement of an existing execution plan, it reuses the plan. This reduces the relative performance benefit of precompiling stored procedures by extending execution plan reuse to all SQL statements.*

The important thing to note about this is that there is no "compiling" of SPs in the traditional sense. In some environments (Oracle, I am told, and in a number of other databases as well), SPs really are converted to a different programming language (such as C or C++) and compiled to native code. This is not the case in SQL Server, for SPs or ad-hoc SQL. SPs have a slight advantage in that the execution plan can be cached and found a bit more efficiently.

Are there any other performance advantages to SPs? In a number of reasonably rigorous tests, I initially did not see any significant differences in performance. I took a SELECT statement that was pulling from a table using a random ID (generated using the `RAND()` function in SQL Server) and for a single SELECT, there was no difference between an ad-hoc SELECT statement and an SP. I expected this, so I duplicated the statement a number of times (100 times, in fact) and compared the execution times, and again I saw no difference. Even when connecting to a remote server over the Internet, I saw no difference.

After more thought, I realized that in my stored procedures I am often doing complex work, building up a temporary table and SELECTing the result of the stored procedure. Rather than doing the individual SELECT statements, I next tried a test in which I created a temporary table and ran 100 INSERT statements in which I used a SELECT for the values to insert. Finally, I ran a SELECT against the temporary table, and then dropped the temporary table. In this example, I did see a reasonable performance advantage in using an SP to perform the task, on the order of 10:1 when operating on a SQL Server both across the LAN and over the Internet.

So what are the performance benefits of using SPs? If you have complex work to do with the data, it makes good sense to work as close to the data as possible. If you are doing complicated operations on data that cannot be expressed easily using standard SQL set-based operations, it is possible that SPs will be quicker than looping through results on the client side and doing the calculations. Note that the issue of when to use or not use SPs will change when SQL Server 2005 emerges out of beta, since you will be able to use VB.NET and C# to create stored procedures that are genuinely compiled.

Performance: Some SP advantage when complex operations are involved.

The Maintenance Argument

Rob argues that ad-hoc SQL is brittle. He further argues that using SPs enables the data model to change in significant ways without impacting the application, if the SPs isolate the application from those changes.

Frans counters:

> *You can use stored procedures or ad-hoc queries, you have to change the calling code to make sure that column gets a value when a new row is inserted. For Ad-hoc queries, you change the query, and you're set. For stored procedures, you have to change the signature of the stored procedure, since the INSERT/UPDATE procs have to receive a value for the new column. This can break other code targeting the stored procedure as well, which is a severe maintenance issue. A component which generates the SQL on the fly at runtime, doesn't suffer from this: it will for example receive an entity which has to be saved to the database, that entity contains the new field, the SQL is generated and the entity is saved.*

The part of this comment that I vehemently disagree with is that changes to the signature of a stored procedure will break your application. Other comments to Frans' post continue to argue that changes to SPs will break client code. While it is possible that any change to the database will break client code, I would argue that SPs do, in fact, enable you to change the database without having to change the application. Let me give you an example.

I was working on a system that would identify objects using a GUID rather than an integer ID number. This was important because, on occasion, the application would be passing object identifiers in a URL in the ASP.NET version of the application, and using GUIDs ensured that someone could not easily try guessing an alternate value. For instance, if a user is working with OrderID 5, it does not take a master hacker to change the URL so that it points to OrderID 6 to see if anything interesting was available in that other order. Other security methods could be employed to prevent this sort of poking around, but for this project, the GUID method seemed best.

Later, a different application was accessing the same database, and for a number of reasons that system was to be written keeping track of the IDs rather than the GUIDS. I simply added an ID parameter to the SPs (with a default value) and the old and new code coexisted without any breaking changes to existing client code. More important, both applications were going through the same database code, so any rules that needed to be followed when accessing data were applied to both systems.

I follow the same sort of rule that was applied to interfaces in the COM world. When an SP goes into production, no breaking changes can be made to that SP. If there are changes required by some new code that will require signature changes that cannot be overcome using default values for parameters, I simply create a new version. Likewise, I create a new version if there are significant internal changes to an SP that modify what the SP does in a way that will compromise client code. So rather than break spTest, I create a new SP named spTest2.

Frans also argues:

> *Now, let me add something about performance here. Say, you have a database with 100 tables, with an average of 7 fields per table. This requires 100 update stored procedures, at least (the crUd procs). These 100 procedures will have on average 7 parameters to update all fields at once, because you can't create optional update statements in stored procedures.*

This is just not true. By allowing default values for SP parameters, you can properly send only the parameters you need, and use default values and IsNull() to update a table with some values missing.

So how important is it that changes to the database might require changes to client code? Regardless of whether you use ad-hoc SQL or SPs, you may have to change client code at some point. How difficult this is varies greatly. If you have a single server ASP.NET installation, making changes to ad-hoc SQL inside an application or making the changes to an SP may be relatively simple. If you have a cluster of Web servers pointing to a single database server, changing the client ASP.NET code becomes more difficult than changing the SPs on a single SQL Server. And if you are maintaining (as I am) a fat client (or as Microsoft loves to call it, a “Smart Client”) application running on hundreds of desktops across a number of towns or counties, any change required in the client application causes an unacceptable burden.

At a prior employer, I supported an application in which any change meant that literally hundreds of CDs needed to be burned and distributed across the country. Making the changes in SP logic in SQL Server and sending the script changes to the customers to apply centrally is a better solution. Note that one of the improvements in maintaining SQL Server these days is Red Gate Software's SQL Compare. I currently keep two copies of most databases I deal with: one that exactly mirrors the structure of the customers' database, and another that is used to make development changes.

When I want to make the changes live, I use SQL Compare to create the change script between the development server and the copy that is like the live server, and ship the script out to the customer, safely moving the changes to the customers' database.

One wild card here is that in some organizations developers are not allowed to manipulate SPs or other objects on the SQL Server directly. In this case, for non-technical reasons, SPs are not a good option.

As far as maintenance goes, regardless of whether you use ad-hoc SQL or SPs, an unskilled person will create a maintenance nightmare; a skilled practitioner will do a great job. I have also created server-side installs that automatically update clients when the client code needs to change, eliminating the need for every client machine to be touched by someone. One click deployment will also be helpful in this respect, once it becomes widely available.

Maintenance: For me, SPs have a slight edge in convenience.

The cross-platform tie-in argument

The cross-platform tie-in argument

One argument against using SPs is that they tie you to a particular database platform. But is that always true? Neither Rob nor Frans address this issue in the posts I reference.

I know of one developer who created an application that can be used in two modes. The program can be operated on a LAN with all users hitting the same SQL Server which uses SPs extensively), or it can be operated independently, where the database is Microsoft Access (which does not use SPs). The program runs all database access through a known API supported by a pair of pluggable dynamic link libraries. There is added complexity, since changes need to be made to both code bases at the same time. For this application, the tradeoffs were well worth it.

Can an application be too flexible? The answer is yes. If you don't use features of your chosen database platform because you might need to move to a different database someday, you are probably giving up too much. I have built many applications that had to account for the possibility that someday a different database back end might be required. I have never been called to make that retrofit. If multiple back ends are not an initial requirement, I would not worry much about it. I would, of course, partition the application so that database access is handled by classes separate from the presentation logic, but that is a good idea no matter what.

Cross platform tie-in: Ad-hoc SQL has the advantage here, but do you care, and is there a good way to overcome the limitation anyway?

Conclusion

So, should you use SPs or ad-hoc SQL? The answer is “it depends.” I have placed myself firmly on the side of doing all database access through SPs. I do so knowing that I am not getting any unique security benefits using SPs, knowing that the performance benefits are not as clear cut as I once might have thought (but are still real in some cases), knowing how to leverage SPs to minimize the maintenance load, and understanding that I am more tied to SQL Server than I might be if I were to use ad-hoc SQL. What do you think?

###

Doug has a blog post on this article, and comments or questions can be posted there:
http://weblogs.asp.net/dreilly/archive/2005/03/30/396251.aspx

Douglas Reilly is the owner of Access Microsystems Inc., a small software development company specializing in ASP.NET and Mobile development, often using Microsoft SQL Server as a database.

Related Content

- SQL Injection Attacks by Example
- Stored Procedures
- Using ADO.NET with SQL Server