

Putting Stored Procedures - and DBAs - to the Test: Part 1

William Vaughn

Stored procedures form the core of many three-tiered applications. Knowing how best to write, debug and test them is an essential skill for .NET developers—assuming your DBA grants you permission to do so.

It seems like I spent the whole month of June on the road. First, I gave three sessions (and attended many others) at TechEd in Dallas. From there, I went straight to Stockholm to do a keynote and two more sessions for the Nordic conference. After four days at home, I was on a plane to New York to tape a [WatchIT](#) segment on Visual Studio NET. During these trips I had lots of opportunities to talk with developers working on all kinds of projects. While many people wanted to know if they should be incorporating XML in their projects somehow, many others had questions on the role of stored procedures in their designs. And, although I wasn't quite sure if XML fit into their application's architecture, I *did* know that most of their designs should be built around stored procedures—especially given what I know about Yukon (the next version of Microsoft SQL Server).

Why stored procedures?

Microsoft, I, and virtually all of the data access writers I know have been talking about, encouraging, and incorporating stored procedures in their designs and applications for over a decade now—long before Visual Basic 1.0 hit the streets. For SQL Server and other serious DBMS platform developers, stored procedures are a way to get many of the features of centrally located and managed code components that other code reuse mechanisms like DCOM have strived to emulate for some time. SQL Server "knows" how to launch these pre-compiled blocks of code, cache and reuse their query plans, allocate memory and threads for them, manage their transactions, and optimize their use by hundreds to thousands of simultaneous users. Database administrators (DBAs) have always used stored procedures as a primary gating mechanism for controlling access to the base tables, thereby protecting both sensitive data and the relational integrity of the database itself. But I don't propose to further extol the virtues of stored procedures—we all recognize that they're here to stay, and most developers, architects, and DBAs have long since integrated them into their database application toolkits.

Who should be writing stored procedures?

Actually, a lot of factors go into the decision of who should write stored procedures, and there's no simple answer the jury's still out. First, you have to consider the complexity of the development shop. If you're writing code on your own, you certainly don't have the benefits (or hassles) of using a professional DBA to manage the stored procedures for you. However, if you're working in a larger development team, you'll probably feel the need for at least a part-time DBA.

A DBA's roles and duties are wide and varied. I've seen situations where a DBA can make or break a team's productivity - and save or destroy any DBMS. One problem is that Visual Studio NET is so easy to use. Except for managing permissions (which it *still* can't handle), Visual Studio .NET has tools to let any login (with the right permissions) create and alter stored procedures - along with the base tables, functions, and even the data. It even has mechanisms to let anyone to step into, run, or interactively debug stored procedures. The trick here is to balance the need to protect the DBMS (and its stored procedures) and the productivity of the development team. If the developers know how to write good stored procedures (many, but not nearly all of them do), then it *might* (just might) make sense for the DBA to grant permission to a selected role or individual developer-class Login to create stored procedures. This also means that this account must (usually) be granted DRI (declarative referential integrity, i.e. schema) rights to the underlying tables. This makes some DBAs (and development managers) shudder. They don't want just *anybody* (and especially not Sam, that developer over in accounting who doesn't even wear socks) changing base tables whenever the spirit moves them. It's a tough question and it's going to have to be up to your company's management (if any), or *you* to gate access to the keys to your DBMS's kingdom.

Developer vs. Application Accounts

One concept that seems to be missing from the Visual Studio .NET documentation is "how to create and use database connections to develop applications". That is, the connection a developer uses to interface with Visual Studio NET is often (and should often be) different from the connection an application should use to interface with the data. The permissions that a developer account should have are (and often have to be) much broader than the permissions that the application needs to execute the stored procedures called. No, I'm not going to get into using "sa" as an account to develop with. This is just flat wrong. If you don't know why at this point, put this magazine down and step away from the vehicle. The DBA's role here is to create and manage both application Logins and User Ids *and* appropriate *developer* Logins and User Ids. They should be treated as distinct.

Creating developer accounts

When a DBMS is first set up, one of the duties the DBA (or whoever is wearing the DBA hat) has to take on is to create developer (and application) roles or Login Ids. I won't get into the benefits of roles (there are lots of excellent reasons to use roles instead of individual accounts - consult SQL Server's Books Online for more), but suffice it to say that if you have to replace team member Sally with Sam, you might find that Sam balks at logging in with a girl's name. Unfortunately, Visual Studio .NET doesn't include a handy GUI utility to create or manage logins or assign permissions to them. If you're using Microsoft SQL Server, you'll probably want to use SQL Enterprise Manager for this task. (If you're using MSDE and don't have SQL Enterprise Manager, consider springing \$US 49 for your very own copy of the SQL Server Developer Edition You'll spend that on a pair of Levis and a clean shirt nowadays, and it's well worth it to get the entire suite of tools that come with the Developer Edition (and you can wear those jeans another couple of months—big gaping holes are “in”).

So, step-by-step:

- Use SQL Enterprise Manager to create one or more “developer” Login accounts. If you're using integrated security, this is as easy as setting up an NT domain account and adding it as a valid Login in SQL Enterprise Manager. If you're using mixed mode security (which is a lot more work for the DBA), you'll need to add a new Login or Role *and* set a password for it.
- Set the default database for the account—it defaults to “master”, so this is an important step. The default database is the database the account should be using and the database that the system will address by default if the code doesn't mention a database name when it addresses objects.
- Set the permitted database(s) for the account. Without this step, the developer(s) won't be able to open a connection to the database because they won't have permission to log on to the specific database. This automatically adds this account as a User in the target database(s).
- Next, select the target database in SQL Enterprise Manager, right click, and select properties. Choose the Permissions tab and find the developer account you set up. Click on the appropriate boxes to “grant” (check mark), or “deny” (red X) the operations you wish to grant or deny to this account. In Figure 1, I granted permission to create stored procedures to the developer account, but specifically denied permission to Create Table, Create View, or Create Default. Since I didn't specifically grant permission to Create Rule, Create Function, Backup Database, or Backup Log (off the screen to the right), this account will not be permitted to perform these operations.

Figure 1 Setting database operation permissions.

Setting Permissions on the base table(s)

No, you're not done with SQL Enterprise Manager quite yet. Next, you need to drill into the target database this developer account is supposed to be accessing and grant permission on *specific* base table(s) that the developer needs to query, update, or alter. This is done by drilling into the User list in SQL Enterprise Manager and right clicking. Select “All tasks” and “Manage Permissions”. The dialog (shown in Figure 2) lists the objects in the database and permits you to select those SQL operations you want the account to be able to use. In this case, I granted all rights to two of the base tables in the Biblio database. When the developer creates a “connection” using the Server Explorer in Visual Studio. NET, only these “checked” tables will be made visible. Unless you check “DRI”, the developer won't be able to change the schema of the table. (This is a good combination of settings if the DBA wants to maintain control of the database but still let developers see the data schema to use it in a query or to let Query Builder see the data tables to generate SQL queries.)

Figure 2 Granting permission on base table(s).

Granting permission to debug stored procedures

No, you're not quite done with SQL Enterprise Manager. If your developers want to step through or debug T-SQL stored procedures, you (or the DBA) will need to grant permission on the Extended Stored Procedure, *sp_sdidebug*. This can only be done using a login account with rights to change the Master database. Drill down into the Master database using SQL Enterprise Manager, examine the list of Extended Stored Procedures. Find “sp_sdidebug”, and right click. Select “Properties” and “Permissions”. Notice that your “developer” account might not be listed here—only SQL Server users, roles, and “Public” that are granted access to Master are shown. You'll have to decide whether to grant your developer account access to Master - or grant permission to Public to permit developers (and any “public” account) rights to do T-SQL debugging.

Next month, we'll continue probing this topic. I'll show you how to call your stored procedure - and to debug it.

LINK WWW.BETAV.COM

bio on file.

