

Re: Sql / Dot Net General Discussion

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.general/2006-10/msg00659.html>

- *From:* "Dave Sexton" <[dave@jwa\[remove.this\]online.com](mailto:dave@jwa[remove.this]online.com)>
 - *Date:* Sat, 21 Oct 2006 23:05:01 -0400
-

Hi Doug,

My primary experience is developing applications using VB or DotNet. I have some sql skills but they are limited. In a previous company our concept on SQL was that it was used for very simple work, (i.e. insert, update, delete, select, etc). The applications we wrote did the bulk of the work. We had very limited DTS's wrtten and stored procs were very small.

I feel the same way as Tom on this point.

In the company I've been working at for the last year, we have two different mindsets on this issue (to be honest, the numbers of people who feel like I do have dwindled, we've lost some of our DotNet developers in the last few months). A very large amount of work is being developed in SQL and even where our DotNet applications are concerned, I have seen some push for putting a lot of the work into the stored procs instead of having them be simplistic like I mentioned in the previous paragraph. I have seen stored procs called by dot net apps that call other stored procs, that call others, etc. Some of these procs are like minature apps in of themselves.

Again, I think Tom's on point here as well.

I have a hard time wrapping my brain around why anyone would do this. I believe that this type of design is problematic for maintainence at the very least.

If you are primarily a .NET developer, then it might be a maintenance problem to enforce, in a database, a large number of business rules that are unrelated to the entegrity of data. But if you are a DBA, then implementing any business logic in managed code might be a maintenance nightmare for you.

Re: Sql / Dot Net General Discussion

Normally you have to be more concerned with the logic that is handled in the database, as a code developer, than a DBA has to be concerned with the logic written in managed code. This is because developers have to use the data from the database in code, and so must be aware of the schema, format, data types, constraints and any business rules that will be enforced that might affect the known state of the data. DBAs need be concerned with connections to the database and when, how and how often they will be used, but not necessarily how the data will be used within the application. Of course, these ideas apply only to segregated development scenerios where sticking to ones own tier is enforced, but that is usually desireable if you have the means. i.e., DBAs should concentrate on meeting the data requirements of the solution in terms of performance, scalabilty and data integrity. They must be concerened with when, how and how often the database will be queried but should have little concern for what the application will do with the data that is returned, effectively seperating the tiers and providing for more focused management of one particular facet of the solution – the data. The business rules that DBAs enforce in the database should, therefore, be directly related to the integrity of the data, the quantity of the data, how/when the data will be used and how often it will be used.

My point is that certain business rules must be enforced in the RDBMS, such as those that help to define the entities, their relationships and the required domain integrity. If they aren't enforced in the RDBMS, then you might as well just use a flat-file ;)

If your DBAs are using triggers, procedures, functions, rules, data types and constraints to ensure the integrity of data according to the business requirements of the solution, then I think they are just doing their jobs. If they are doing anything less or anything more than I think you should question their motives (staff meeting :)

Requirements that aren't directly related to the integrity of data should really be enforced in manage code. For one thing OOP is obviously more robust and scalable for a number of reasons, so why enforce business rules that have no relationship to the integrity of data in a database language such as T-SQL? The common answer, as Tom stated, is that DBAs can read T-SQL but can't read C#, for example. As a developer, you might have a hard time with T-SQL and prefer C#. In an enterprise development scenerio, however, I think the ideal situation is to use T-SQL for CRUD procedures and the enforcement of business rules and requirements that are directly related to the maintenance of data integrity at the entity, referential and domain levels, and to define any other business requirements in managed code, whenever possible.

As an architect, use your own discession in choosing where to enforce particular business rules that idle on that fine line between data-integrity requirements and data-manipulation requirements, and remember that you can and usually should enforce those requirements in managed code unless they neatly fit into a trigger, IMO.

Of course we don't live in an ideal world and so I will acknowledge exceptions

Re: Sql / Dot Net General Discussion

to the thoughts I laid out above. Financial applications, for example, commonly have mathematical requirements that one can benefit from using SQL Server's built-in formulas for standard deviation and even more complex operations, especially because they are designed to be used in aggregation and use sets as input. The database is obviously well-equipped to handle such requirements that aren't related to the integrity of data. However, other than certain mathematical requirements I'm having a hard time figuring out good reasons to enforce business rules in the database. And executing mathematical formulas in the database isn't always the best choice anyway, but it will never hurt to implement it in managed code.

But I would think it puts an unnecessary burden on SQL too. I just don't know how to prove it.

You can't really burden an SQL Server with business rules if they are rules that enforce or maintain entity, domain or referential data integrity because that's what an RDBMS does. I think other rules should normally be enforced in code, as I mentioned above. Certain rules not related to data at all could also be implemented in the database when automation through scheduling is required. Enforcing these rules within SQL Server is not a burden, especially since SQL Server provides robust support for this secondary functionality.

In SQL Server 2005 you can even execute managed code, so the lines are beginning to blur anyway ;)

When I've mentioned this, some of the feedback I get is that my concept would cause network traffic that is unnecessary (i.e. multiple stored proc calls, etc). Again, I am unsure how to test/verify such a claim.

Well you shouldn't need to test that comment at all. If the database is processing most of the business rules of your application, the only calls that you'll need to make are those for CRUD operations (ideally). Instead, if the application is enforcing most of the business rules then the application will most likely have to make extra calls to the database to enforce constraints, format data and will definitely have to update multiple tables for reasons other than simple CRUD operations. Yes, the latter will probably increase network traffic greatly and that's why the RDBMS aspects of SQL Server should be used to serve their purpose, reducing the amount of work the application has to perform when enforcing data-centric business constraints and requirements; however, I don't think that is a good excuse at all for implementing business requirements that aren't related directly to the integrity of the data, in the database. By doing so you may be reducing network traffic but you are certainly creating more work by not leveraging the benefits of an OOP environment and will definitely not be able to address all of the business requirements in the RDBMS alone, creating an illogical separation of certain rules making it confusing to figure out or remember

Re: Sql / Dot Net General Discussion

whether a rule was implemented in the database or in managed code, or even where it may have been implemented in the database itself, such as triggers, multiple procedures or functions. It's much easier, IMO, to find a business rule in a self-titled business entity object than to search through a list of 155 stored procedures with a terrible naming convention (let's face it, no matter what naming convention you choose it's always terrible when you have 155 or more stored procedures listed sequentially in a TreeView ;)

I would think the best approach is to have your business logic stay in DotNet if you have a DotNet app. Obviously if you have a process that doesn't get put into an application at all, then I believe the logic should stay in SQL. I do question the necessity of having that type of work happen as often as I'm seeing it though. DotNet can be used to write pretty much any type of application that you would do in SQL.

I hope I've made my opinion on this clear :)

<snip>

Dave Sexton

.