

In This Issue

- John continues discussion of failure rates in ERP
- Stu discusses similarities between pharmaceuticals and software development
- Wally does a performance analysis of TSql and SQL CLR

Why Software Projects Go Astray

Part II

By John Croft

Studies show the failure rate of ERP (enterprise resource planning) software project implementations to be 61%, with even higher rates of failure for other types of software. In Part I of *Why Software Projects Go Astray*, we compared a software project to building a house, focusing on the problem of unmatched goals between the client and the developers. Let's continue.

Next, there's the problem of estimation. The contractor can measure the drawings to determine the required amount of materials to build a house. What does the developer have to measure?

See *Software Projects* on Page 3

Software and Pharmaceuticals

By Stu Lustman

What do these two industries have in common? On the surface, not much. However, a look just below the surface finds something else. Both of these industries are front loaded, in terms of expenses. In the pharmaceutical trade this is known as R&D (research and development).

See *Striking Similarities* on Page 3

TSql vs. SQL CLR: Performance Analysis

By Wallace "Wally" McClure

The subject of one of my recent .NET User Group talks was Sql Server 2005 CLR Objects. During the talk, I was asked, "Which should I do? TSql or CLR objects?" Personally, my generic answer has always been that if a developer can perform something in TSql, its best to do it in TSql. If you can perform the necessary operations using TSql, the performance will be better in general. This is before one takes into account the project overhead of managing the source code for stored procedures, triggers and other database objects.

Once you decide that your scenario actually needs a CLR Object, what's the best way to perform the operation? Should this be done entirely within a CLR Object? I want to take a look at these questions based on the WebSearch spider and its use of CLR Objects to perform certain data operations.

Question: What is the best way to implement a CLR Object within a Sql Server 2005 Database? Which will perform the best?¹

Short Answer: Do as much as you possibly can in TSql. Use CLR Objects judiciously. Once the need is determined, perform as many operations in TSql and use CLR Objects only as absolutely needed.

Much Longer Answer: First off, not all situations are the same. In the following example, we are looking at this from the standpoint of keeping data synchronized within a single database table.

See *TSql* on Page 2

¹ Terms like "best" are highly subjective. As a result, the question that was answered is which *performs* the best. Obviously, that type of question is very dependent on the scenario. As a result, please test this in your own shop to get the best results for your application.

TSql

To answer the question, I have built four different solutions to this problem:

- Mostly TSql. A TSQL trigger which creates a cursor and calls out to some custom CLR Functions on an individual basis.
- TSql Cursor. A TSQL trigger that creates a cursor that calls out to some custom CLR Functions through a look within the trigger.
- A CLR Trigger which uses a Generic List <String> to hold the pending data along with a SqlDataReader to iterate through data. Afterwards, each url is updated by iterating through the Generic List.
- A CLR Trigger which uses a DataSet/DataTable to hold the data and then the Sql Data Adapter's .Update method to transfer the data back.

Measured relative performance of each		
	Average time	Standard Deviation
Mostly TSql (base)	1x	10 / 1x
TSql Cursor	1.42x	.61x
CLR with Generic List	2.04x	10.4x
CLR with Datasets	4.65x	15.6x

Performance Analysis: Now that we have seen the performance differences of the four solutions, let's try and analyze the differences between them.

The "Mostly TSql" solution performs the best. This is not a surprise. TSql has been around for a number of years. It has been embedded within SQL Server and has a highly optimized relationship with the database. While embedding the CLR with the database, it provides its own overhead, so it's not a shock that the "Mostly TSql" solution performs the best.

The TSql Cursor solution takes 1.42 times as long as the "Mostly TSql" solution to perform the same work. This is not surprising, as cursors generally take longer to perform a given operation compared to comparable sql commands.

The real surprise in this test is the performance of the CLR trigger, which used a Generic List to hold its sql commands that are to be sent back to the database. The Generic List Trigger performed more than twice as fast as the Dataset Trigger. Personally, I would think that the Dataset would be more optimized in this scenario; however, it does not perform as well. This is probably due to much more overhead, as opposed to a 'simpler' Generic List.

BobB² suggested this reason for the Dataset not performing as well: "Arrays (especially multi-dimensional arrays) *might* be faster if you're doing array-only processing, preferable (sometimes) over gobs of temp tables. But using a heavyweight object like a DataSet takes memory management away from the SQL engine. SQL engine is always best when it is managing all the data buffers."

How was the data created? It is important to communicate how the above data was calculated.

1. In each situation, approximately 15 updates were performed against the 1000 records in the table. The command is "UPDATE TBLSEARCHRESULTS SET SERVERNAME=NULL"
2. When testing was performed, the triggers that were not being tested were not installed on the database table.
3. The first 5 updates were not used for the calculation. This is due to the necessary startup and JIT operations that are necessary for the objects within the database. The idea is to simulate a running system. As a result, startup isn't a major concern for this test.
4. The sql commands AVG() and STDEV() were used to create the data in the above table.

The test platform for the scenario was:

- Laptop.
- 1.8 GHz Pentium 4.
- 1 gigabyte of RAM.
- 60 gigabyte hard drive.
- Windows 2003 Server with Service Pak 1 and up to date security updates through Windows Update.
- Sql Server 2005 Service Pak 1.

Areas of concern/problems. There are a number of issues and assumptions made within this test that could cause problems with the data that was generated:

- There is no need to go outside of the single database. If there was a need to go outside of the database, this test would not be applicable to your needs.
- Only updates are tracked. No inserts were tracked during this testing.
- This was tested using a laptop, not a multi-processor database server with an appropriate drive system layout. Different results for different hardware layout would not be unexpected.
- I don't believe I have any Sql Injection attack openings, but I might be wrong.

² [Bob Beauchemin](#) took a look at this before I published it. Many thanks to him for taking the time to look at it and provide a couple of suggestions and insight.

Software Projects

The developer has to estimate how much code will have to be written to do all of the things on the list, but the reality is that the developer won't really know how much code has to be written until it's finished. Then, there is the 'how' list. There isn't much question about how to paint a wall or install flooring but, for the developer, each item on the to-do list requires a unique solution. And it gets worse. If the painter paints a room the wrong color, the doors on the house will still work. But if a developer solves one of his problems the wrong way, the whole system may fail.

Summarizing, the developer starts with an incomplete or inaccurate list of action items that the customer can't verify, from which is built an unverifiable list of coding assignments which will be assembled in a manner that will be determined at the moment they are created.

If this sounds like a recipe for disaster, then you can understand why ERP implementations fail 61% of the time and fully custom software projects fail at rates approaching 90%.

What can be done about this, and why do some projects actually succeed? The reality is that many aspects of success are completely dependent on the developers. The developers will be responsible for the coding, the 'how' part of the project. No matter what resources are put into planning, if the code is bad, the software won't work. If good code is written with the wrong focus, the software won't work.

For a software project to succeed, the right code has to be written to solve the right problem, and that requires good programming talent. From the client side, the right projects need to be chosen. Many clients would like to do everything now, but that simply can't happen. Clients should pick projects that are manageable in size with understandable results. When one goal is achieved, then consider the next goal. Don't look for big expenses to produce some big future return. The larger the project, the more likely it is to fail. Software infrastructure should be a series of value investments made over time that result in ever more functionality while providing immediate benefits to some identifiable group of users.

“For a software project to succeed, the right code has to be written to solve the right problem, and that requires good programming talent.”

Striking Similarities

That includes all the time and money the drug companies must put out up front just to create the drug that helps with high blood pressure or enlarged prostates. In the software trade this is sometimes known as 'development time.' Development time is everything from a needs and requirements analysis to programming to a completed package that is deliverable to the client. Sounds kind of similar doesn't it?

The other important idea these things have in common is scalability. Due to development costs (development time or R&D time); both of these industries are very heavy in front loaded costs and fixed costs. For our pharmaceutical company, this is evident in the fact that creating the first Viagra pill cost \$1 billion (all in R&D costs). However, the second one only cost about ten cents, and the billionth one still only costs about ten cents, as the \$1 billion development cost is spread out over every future unit of the drug that is created.

A good database is scalable, as is all good software generally. With custom software, like we create here at Scalable Development, the software company isn't the one that receives the benefit of the scale (like Pfizer does from Viagra). The client company that purchases our custom software gets the benefits of scale. They get a database that allows them to scale up their business whether only they, themselves, use the software, or they have 50 employees that also use it, or 5000. This is a tremendous benefit to the client, as the software is paid for once up front. So how much does it cost to bring in an additional user in the form of an additional employee? ZERO. This puts our client in the position of giving their employees the tools they need as early and as cheaply as possible so that they can start generating revenues to justify their salary. The same applies when the client continues to grow and has to open another branch office in another city. In fact, a scalable database costs less for the client to move it into a new office than the furniture does. Furniture doesn't do anything to help our clients make money but, thankfully, our software does.

Scalable Development, Inc.

Atlanta Office

2900 Chamblee Tucker Rd.
Building 5
Atlanta, GA 30341
770-458-6590

Knoxville Office

118 Durwood Rd.
Knoxville, TN 37922
865-693-3004

www.scalabledevelopment.com