



TECHNICAL PAPER

Synchronizing databases

with the SQL Comparison SDK

By Doug Reilly

redgate
ingeniously simple

In the wired world, our data can be almost anywhere. The problem is often getting it from here to there.

A common problem, that I have been asked to solve by a number of clients, is moving data and database structure changes from a database located on the customer's intranet up to the database server on the internet. The companies involved had essentially static databases on the live internet site and a local copy of the database on which internal users made changes. Even if it were reasonable to do, the customers did not want work to immediately appear on their public internet site. In some cases, the lack of instant synchronization would be a bug. For these clients, however, it was a feature: internal users could try out changes to the database and know that nothing they were doing would become live until an overnight process ran.

Replication methods

The first time this problem arose with a client, I went to SQL Server Books Online and researched SQL Server replication, thinking it would be the solution to all of my problems. SQL Server replication can even replicate with non-SQL Server data sources, something that might be critical for some users.

There are a number of types of SQL Server replication:

Snapshot replication – This is done on a scheduled basis, and makes the target database identical to the source database.

Transactional replication – Subscribers are first synchronized with the publisher and as data is modified the transactions are captured and sent to the client. This sort of replication requires a relatively stable and consistent connection.

Merge replication – Multiple subscribers merge data back to the publisher, operating autonomously. Because changes can occur in multiple places, there can be conflicts and there needs to be some allowance for conflict resolution.

Since the clients would only be changing data in a single location, merge replication was clearly overkill. Transactional replication was also more than was needed, since the client would never directly change the database on the internet server. Snapshot replication seemed to be a solution for me. But, not so fast.

Replication can require changes to the database and creating a place for SQL Server to drop temporary files. An extra column is added to the database to support some types of replication.

In addition to potential database changes, SQL Server replication can force you to make security changes so that replication can continue. This is not a lot of fun, and sometimes is not even possible on databases where you do not have complete control over the database.

Data Transformation Services (DTS)

My next thought was to use the Data Transformation Services (DTS). DTS is a tool often used by developers and database administrators to move data from place to place, and it can also be used to move objects (the structure of tables and views, as well as stored procedures). DTS is very good at doing ad-hoc transfers of data from machine to machine but, as I would soon discover, using it in a production environment is fraught with difficulties.

One of the things that I presumed would work automatically was some sort of transaction processing. That is, I thought that either the entire batch would succeed or none of it would succeed. That was not the case. In the end, I was able to get the system working through clever use of backups and ON SUCCESS and ON ERROR processing inside DTS.

For my first client, I used DTS to do the nightly database transfer. This required that I backup the destination database, in case it all went badly. It also meant that the site would be unavailable for a few minutes, which was acceptable because the process ran overnight. The transfer went reasonably well, but tweaking security and creating the necessary folders and "backup devices" in SQL Server required a reasonable amount of tinkering. The solution was also fragile: It occasionally broke when the DTS packages were saved, and always broke when the source or destination server needed to change.

A better solution: the SQL Comparison SDK

Shortly after my first experience with synchronizing a read-only internet database with a read/write intranet database, I started using Red Gate's SQL Compare. For a developer who works offsite, and especially one who needs to create a script to enable a DBA or system administrator to make coding and data structure changes to a live database, SQL Compare does a wonderful job of scripting differences between two databases. I used Red Gate's SQL Data Compare very infrequently, but when the next client approached me with the same problem, I decided to look into automating SQL Compare and SQL Data Compare with Red Gate's SQL Comparison SDK.

There are two options within SQL Comparison SDK for automating SQL Compare and SQL Data Compare. You can use the command line to pass in commands, or you can automate the process inside a .NET program written in either C# or VB.NET. If I were a DBA and not a programmer, the command line option would be very attractive. As a programmer, I found the programming interface much more appealing. Rather than relying on return codes from command line tools, I can completely integrate the tools inside SQL Compare into my program.

Since the customer might periodically make changes to the structure of the database schema as well as to the data, my synchronization code has two phases. First, I compare and synchronize the schemas. Then I synchronize the data. Note that the synchronization of data might take quite some time. Fortunately, there is a status event handler on a number of the objects (most notably, on the Utils class) that enables your program to provide feedback on the progress of the operation.

You can download the entire Visual Studio 2003 project. To get the Red Gate objects working in Visual Studio (with complete IntelliSense), I added references to RedGate.SQL.Common, RedGate.SqlCompare.Engine and RedGate.SqlDataCompare.Engine. Adding these references and using the Red Gate objects gives you access to a rich set of properties that have utility far beyond just data and schema synchronization.

Note that this is a sample Windows program, and not exactly what I use to synchronize production databases. The Windows program simply provides a more visual way to follow the program in operation. In the real world, I place the code inside a Windows Service running on the source database machine. The beauty of placing the code on the server inside the firewall is that, except for the need to open up the correct port on the firewall, there are no changes required on the server.

First, I define and declare a structure to contain all the information required to connect to the database:

```
public struct SqlServerInfo
{
    public string server;
public string dbName;
public string userName;
public string password;
public bool useIntegratedAuth;
}
```

When you connect to a database using the Red Gate objects, there is an overload that accepts a username and password, as well as one that presumes integrated authentication. In the sample project, I set `useIntegratedAuth` to true for both the source and destination instance of `SqlServerInfo`. All required information is set in the constructor of the form.

The text box shows details of the current activity, as well as the script used to synchronize the database. The progress bar displays how the current operation has advanced. The button, of course, starts the operation.

The button click event handler calls the `DoSynch` method of the form class. This is the heart of the program. Once the database objects are declared and opened, the code determines the differences between the two databases.

```
Differences diff=db.CompareWith(dbDest,Options.Default);
```

```
foreach ( Difference d in diff )
{
    this.textBox1.Text+=d.Type.ToString() + " " +
        d.DatabaseObjectType.ToString() + " " +
        d.Name + System.Environment.NewLine;
    Application.DoEvents();
}
```

This is a very cool part of the SQL Comparison SDK. While my goal was to automate the synchronization of two databases, I could just as easily automate comparisons of databases, an equally important job. In the example, I display the differences, which, given the code above, look something like this:

```
OnlyIn1 Table [dbo].[UBNormalizedData]
Equal Function [dbo].[udfConvertStringToDate]
```

These lines indicate that the table UBNormalizedData is only in database 1 (the Source database) and the function udfConvertStringToDate is equal in both databases.

Once the differences in schema are known, the schemas can be synchronized. The code that does that appears below:

```
Work w=new Work();
w.BuildFromDifferences(diff,Options.Default, true);
this.textBox1.Text+=w.ExecutionBlock +
    System.Environment.NewLine;

    Utils u=new Utils();
this.lblCurrentActivity.Text="Synchronizing Schemas";
u.Status+=new StatusEventHandler(StatusHandler);
if ( this.dbDestInfo.useIntegratedAuth==true )
{
    u.ExecuteBlock(
        w.ExecutionBlock, this.dbDestInfo.server,
dbDestInfo.dbName);
}
else
{
    u.ExecuteBlock(
        w.ExecutionBlock, this.dbDestInfo.server,
dbDestInfo.dbName, false, dbDestInfo.userName,dbDestInfo.
password);
}
```

This part of the process is fairly involved, with more bits of code than I would like. First, I create a Work object, and from that object I create an execution block. I then create a Utils object that executes the execution block. Note that I also display the ExecutionBlock property of the Work object, again, for demonstration purposes. Anyone who has used SQL Compare will immediately recognize the SQL produced by the SQL Comparison SDK. It is extensively documented, and is designed to either work or fail.

I also set the Status event handler of the Utils instance. The StatusHandler code is very simple, and is shown below:

```
public void StatusHandler(
    object sender, StatusEventArgs e)
{
    if ( e.Message!=null && e.Message!= string.Empty )
    {
        this.textBox1.Text+=e.Message +
            System.Environment.NewLine;
    }
    if ( e.Percentage>=0 )
    {
        this.progressBar1.Value=e.Percentage;
    }
    Application.DoEvents();
}
```

Of course, this code would be very different if this were a Windows Service rather than a Windows program.

Now that the databases have the same structure, we can move on to data synchronization. Note that many of the classes in the next section of code have the same names as the previous objects (for instance, Database). The difference is that the prior code contained objects belonging to the RedGate.SqlCompare.Engine namespace, whereas many of the same named objects in the subsequent code are in the RedGate.SQLDataCompare.Engine namespace.

After opening databases (using a slightly different procedure than that used to open the databases in the SQLCompare namespace), the following code synchronizes the data in the databases:

```
        foreach (RedGate.SQLDataCompare.Engine.Table table
                in commonTables)
        {
            settings.Add( new
                TableComparisonSetting(table.FullyQualifiedName,
table.Fields, table.PrimaryKey.Fields));
        }

        sess.CompareDatabases(dbData,dbDataDest,settings);

        foreach (TableDifference difference in sess.
TableDifferences)
        {
            difference.Selected= true;
        }
        ExecutionBlock block=provider.GetMigrationSQL(sess, true);

        this.lblCurrentActivity.Text="Synchronizing Data";
        u.Status+=new StatusEventHandler(StatusHandler);
        if ( this.dbDestInfo.useIntegratedAuth==true )
        {
            u.ExecuteBlock(block, this.dbDestInfo.server,
                dbDestInfo.dbName);
        }
        else
        {
            u.ExecuteBlock(block, this.dbDestInfo.server,
                dbDestInfo.dbName, false,
dbDestInfo.userName,dbDestInfo.password);
        }
    }
```

Again, this code is rather involved, with more objects than I would like. After a Settings object is created, I add information on each table in a foreach loop, then do the comparison using the CompareDatabases method. I select all differences, get an execution block, and then execute it in the same way as the database synchronization code was executed.

The tool is powerful and enables me to control exactly how my database is synchronized, while ensuring consistent results and the least amount of downtime.

Things to look out for

Of course, no tool works miracles. I was trying to synchronize two example databases, and the source database had one table with an additional column. The added column did not have a default, and did not allow NULLs. Since there were existing rows in the table, the database structure could not be modified to allow this column to be added, as there were no values for that added column. Not surprisingly, I was unable to synchronize the database. This is a fundamental rule of the database, and of course SQL Compare cannot overcome that.

Another possible "gotcha" is that the models of the SQLCompare and the SQLDataCompare namespaces are not as similar as they could be. This is most noticeable when you want to create and open a database in each namespace. The SQLCompare namespace seems a bit more straightforward, with a `.Register` method on the Database object. In the SQLDataCompare namespace, you create a provider and then call `.GetDatabase` on the provider object. Looking at the code in my example, the code to compare the structure of a database is a little different from the code used to compare the data in that database. Both models are fine, but I would prefer using the same model for both.

There is an old saying: "It is not how the elephant dances, but that the elephant dances at all." Even if there is some awkwardness in the object model exposed, the fact of the matter is that the SQL Comparison SDK enables you to do things that are just not possible using any other tool I can think of.

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. He can be reached at doug@accessmicrosystems.net.

Doug has a blog post on this article, and comments or questions can be posted there: <http://weblogs.asp.net/dreilly/archive/2005/03/18/395097.aspx>

Try the SQL Comparison SDK free: www.red-gate.com/sql-comparison-sdk