Where is the truth?

WHITE PAPER SERIES: PART 1 OF 3

DATABASE DEVELOPMENT & APPLICATION LIFECYCLE MANAGEMENT

Dan Wood, ALM Consultant

NORTHWEST CADENCE | WWW.NWCADENCE.COM

Where is the truth?

The Challenge

No, it's not a philosophical question; it's a development question. Developers rightly depend on software configuration management systems to effectively promote code through a set of gates prior to deploying it to production. The "truth" of the application is known. It is authoritative and managed. Many organizations are learning how to effectively promote and deploy through automated processes. Then comes the database.

Very few organizations have an effective – and sustainable – system for managing database change. Of those that do, very few of them have managed to automate the process. Databases and database development endeavors are largely still in the Wild West of configuration management, with manual deployment changes being made to production that may or not be reflected in source control and no clear definitive declaration of the truth. There are a number of reasons for this, ranging from clumsy or ineffective tools to old school intransigent database administrators who insist on personally validating and implementing all database modifications. Whatever the reasons, chances are they can be overcome.

Application Development and Databases: a 3-Part White Paper Series

This series steps through the processes that support an integrated development environment. Pitfalls and good practices will be discussed. So will strategies for deployment and the dreaded deployment rollback. The first paper covers the database development environment, adding a database project to a solution, and discusses the methods to bring database and application development efforts in to harmony. The second covers strategies for deploying the complete solution to various environments for testing and validation. The third paper ends with automated deployment and release management strategies for database projects as an integral part of a full application release.

Part 1: Gathering in the Strays; Bringing Database Development into the ALM Fold
Part 2: Living in Harmony; Integrated Application and Database Deployments
Part 3: The Dream Realized; Automated Deployment and Release Management of Databases

Solution Overview

Business Scenario

In this white paper, Dan Wood explains the way to bring the database in to the solution. It isn't difficult. What *is* difficult is maintaining the discipline.

There are years of habits that need to be unlearned. Not necessarily bad habits, because for many years, there were very few realistic alternatives to the "Cowboy" approach to database development and modification. That was then. Today there are many alternatives and they are all better than letting the database go its own way, because like a sheep, it will go astray if left to itself. Do yourself a favor. Gather in the sheep and pen them up in source control.

Business Results

An integrated development environment whereby the database and application are developed together, tested together, and deployed together. Successfully.

Partner Overview

Northwest Cadence is a national leader in Microsoft Application Lifecycle Management and Software Process solutions. Northwest Cadence provides training and consulting services for clients across the globe on Microsoft Application Lifecycle Management Tools and on Software Development Processes.

Watch Dan's video on Application Development and Database at www.nwcadence.com

Website www.nwcadence.com
Team Blog blog.nwcadence.com
Services clientservices@nwcadence.com

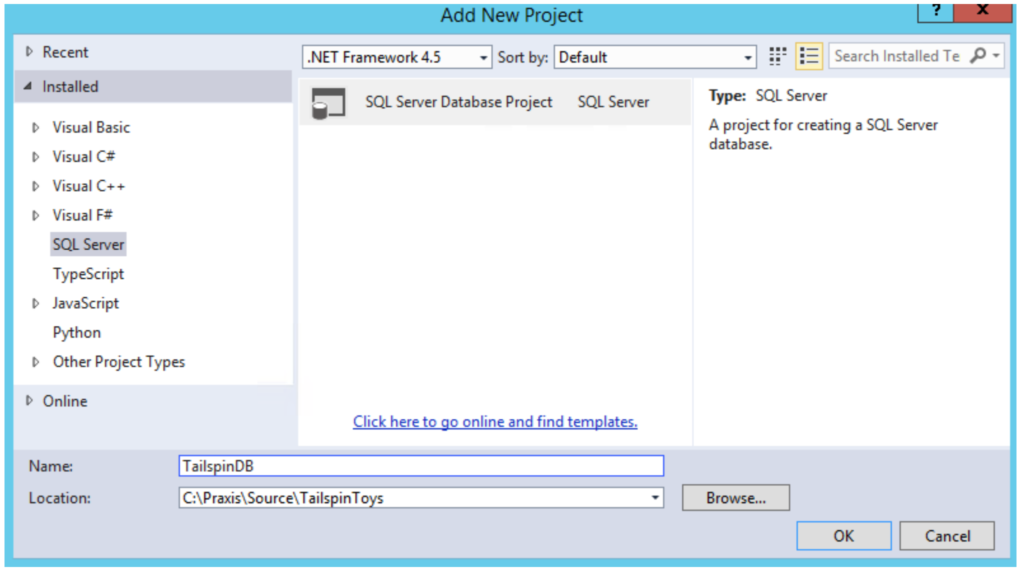
The Solution: Gathering in the Strays

The vast majority of applications being built today are tied to a database in some form or another. Processes for managing the changes to an application are well known and well documented. Managing the changes that occur at the data tier in support of the application changes is less known, and the strategies for handling the synchronization of application and data tier changes are varied; they shouldn't be. If an organization has an established branching strategy for their application, the same strategy should be applied to the database that supports the application. The same principle is true for the promotion, release management, and deployment strategies. What's good for the application is good for the database. There are certainly some different exceptional factors that need to be addressed as far as change management is concerned for the data tier, especially when it comes to roll-backs. I will do my best to address those exceptions, but for the most part the processes for the application and data tier should remain aligned.

Step 1: Capture the Database(s)

For the purpose of this article I will be working with a single database and it's one of my favorites, Tailspin Toys. It's simple enough to get the process down and is a good starting point for adding modifications since out of the box it has no programmability objects. As an application it is also a good example since we can develop and deploy the web application with the database, which is the whole point.

The first step in aligning the application and database development efforts is adding the database or databases to the application solution. To do this we will add a new database project to the existing solution.



Sort by: Default

Type: SQL Server

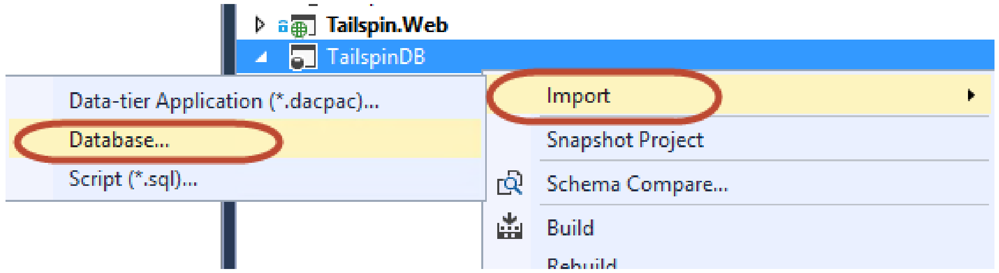
A project for creating a SQL Server database.

Name: TailspinDB

Location: C:\Praxis\Source\TailspinToys

Adding a SQL Server Database Project

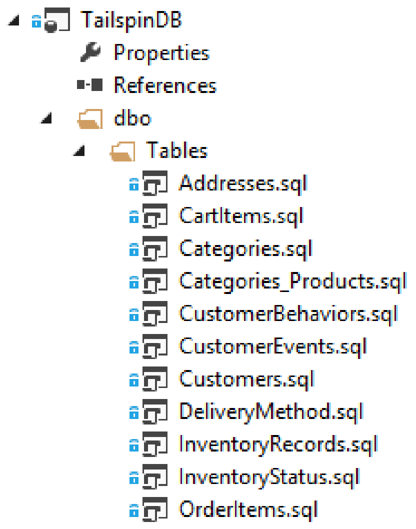
Once we have our new database project, we need to capture the database schema by performing an import of the database. The import will create a set of scripts that can be used to recreate the database.



Importing a Database

Once the import is complete, a set of scripts will be created and placed in the new project.

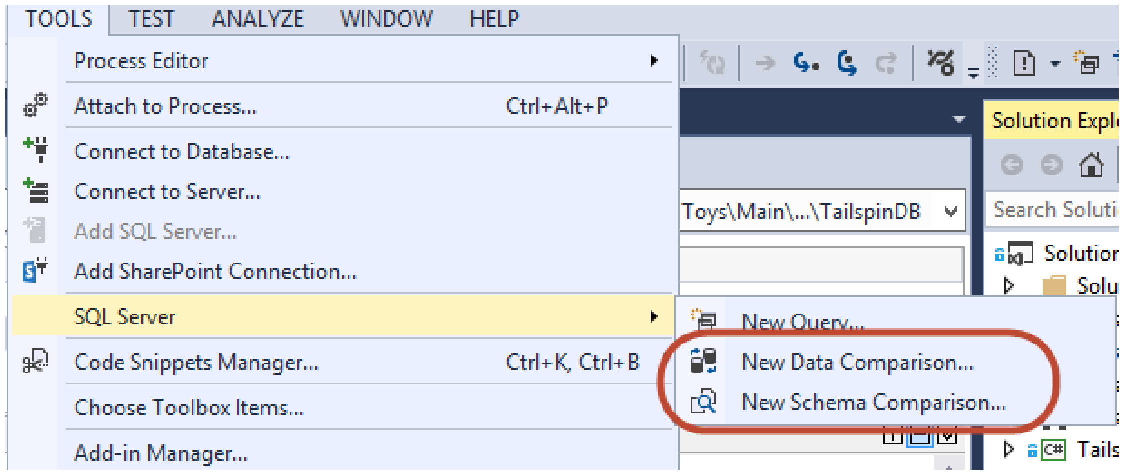
By default the scripts are arranged by schema and then object type. At this point, the Tailspin database consists only of Table objects, but we will be adding different object types as we progress. We now have a project that has been added to our solution that contains (we presume) the authoritative database objects that our project depends on. After adding the new project to source control, the project is available for any other developers to work with as well as for the build engine to use in creating additional instances of the database in various environments.



Checked In and Safe

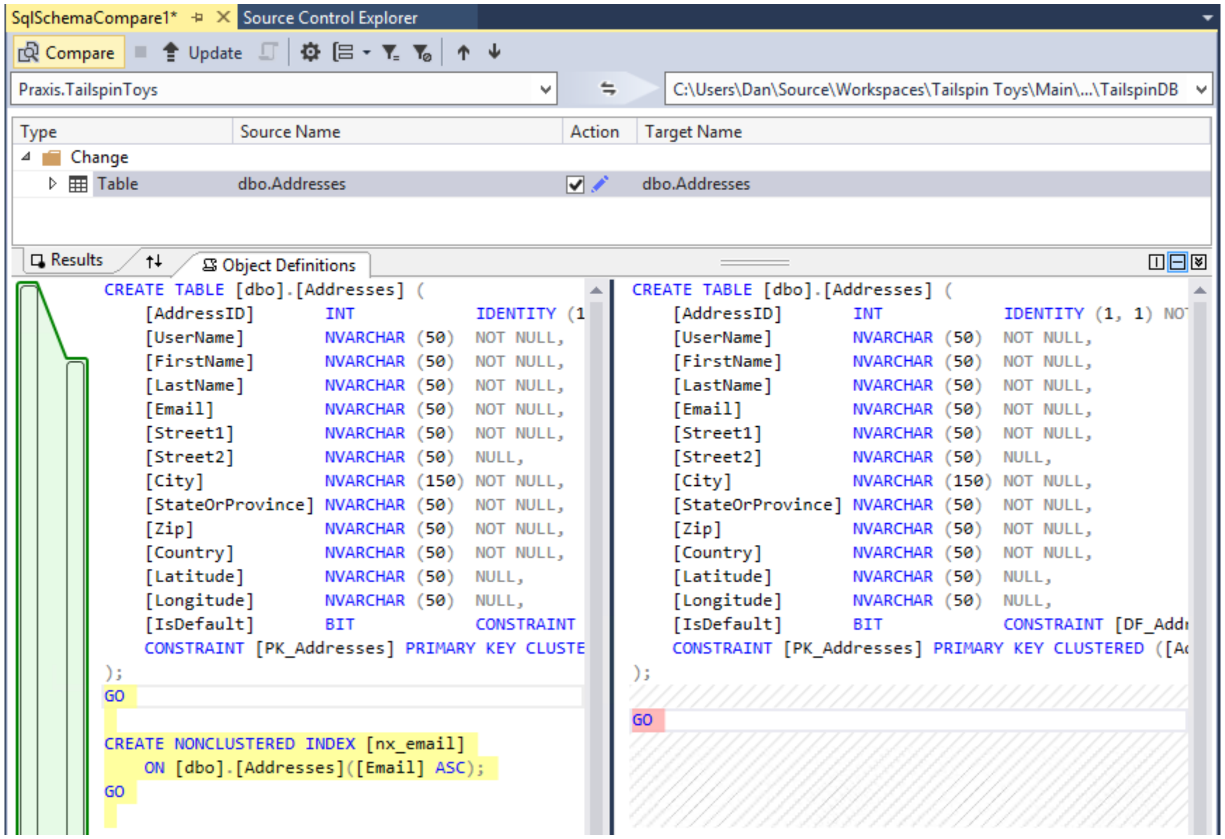
The database artifacts are treated the same as the rest of the solution objects. Modifications to the files - whether done in Visual Studio, SQL Server Management Studio, or even Notepad - will be tracked with Visual Studio's local workspaces. The database project can be branched for isolated development efforts and merged with other changes exactly like application code.

An important discipline to practice and enforce is that all changes to the database be done to the project in source control so that adequate configuration management is applied. However, what happens if that is not the case? What happens when changes are made to a physical database in the development environment? I say development environment because we know that no one would ever make ad-hoc changes to a database in production, right? If the unthinkable happens and changes are made, Visual Studio provides two great tools for finding out what the changes were: The Schema Comparison tool and the Data Comparison tool.



Comparison Tools

With the Schema Comparison tool I can compare the authoritative version (the one in source control), with the physical database where I suspect unauthorized changes have been made. The schema comparison tool analyzes my source, the database, and compares it to my target, the database project. It then outputs any changes to the database much like Visual Studio's code comparison tools.



Type	Source Name	Action	Target Name
Change			
Table	dbo.Addresses	✓	dbo.Addresses

```
CREATE TABLE [dbo].[Addresses] (
    [AddressID]       INT            IDENTITY (1
    [UserName]        NVARCHAR (50)  NOT NULL,
    [FirstName]       NVARCHAR (50)  NOT NULL,
    [LastName]        NVARCHAR (50)  NOT NULL,
    [Email]           NVARCHAR (50)  NOT NULL,
    [Street1]         NVARCHAR (50)  NOT NULL,
    [Street2]         NVARCHAR (50)  NULL,
    [City]            NVARCHAR (150) NOT NULL,
    [StateOrProvince] NVARCHAR (50)  NOT NULL,
    [Zip]             NVARCHAR (50)  NOT NULL,
    [Country]         NVARCHAR (50)  NOT NULL,
    [Latitude]        NVARCHAR (50)  NULL,
    [Longitude]       NVARCHAR (50)  NULL,
    [IsDefault]       BIT            CONSTRAINT
    CONSTRAINT [PK_Addresses] PRIMARY KEY CLUSTE
);
GO

CREATE NONCLUSTERED INDEX [nx_email]
    ON [dbo].[Addresses]([Email] ASC);
GO
```

```
CREATE TABLE [dbo].[Addresses] (
    [AddressID]       INT            IDENTITY (1, 1) NO
    [UserName]        NVARCHAR (50)  NOT NULL,
    [FirstName]       NVARCHAR (50)  NOT NULL,
    [LastName]        NVARCHAR (50)  NOT NULL,
    [Email]           NVARCHAR (50)  NOT NULL,
    [Street1]         NVARCHAR (50)  NOT NULL,
    [Street2]         NVARCHAR (50)  NULL,
    [City]            NVARCHAR (150) NOT NULL,
    [StateOrProvince] NVARCHAR (50)  NOT NULL,
    [Zip]             NVARCHAR (50)  NOT NULL,
    [Country]         NVARCHAR (50)  NOT NULL,
    [Latitude]        NVARCHAR (50)  NULL,
    [Longitude]       NVARCHAR (50)  NULL,
    [IsDefault]       BIT            CONSTRAINT [DF_Addr
    CONSTRAINT [PK_Addresses] PRIMARY KEY CLUSTERED ([A
);

GO
```

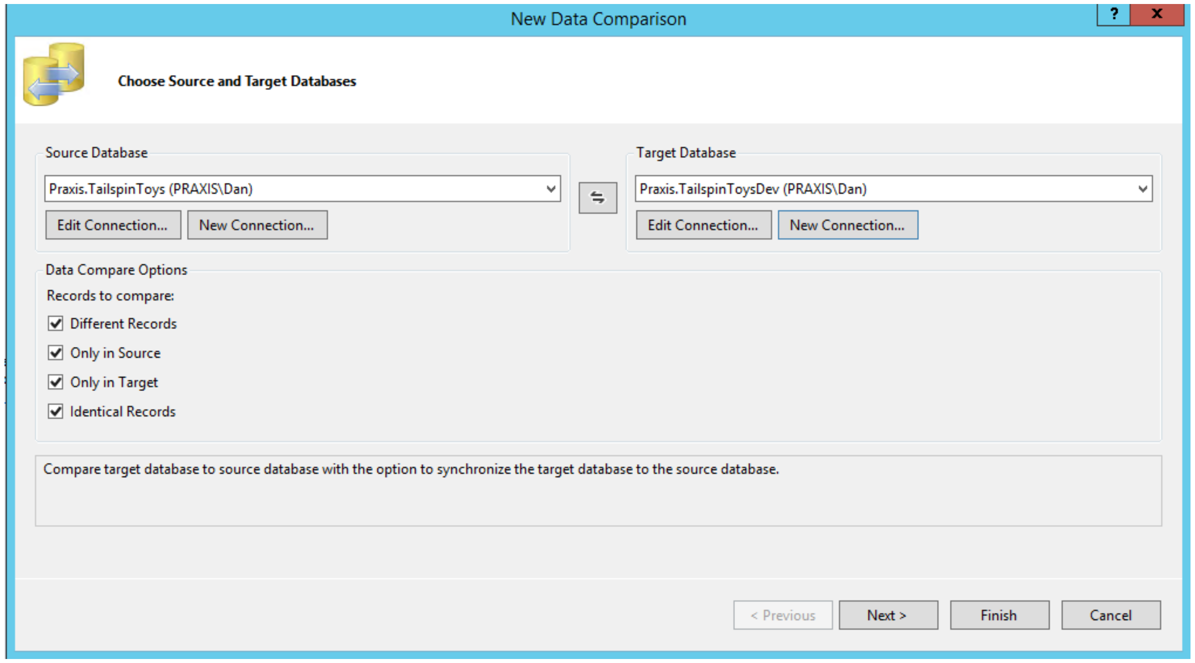
Hmmm, looks like the DBA has been doing some indexing

Is that a good thing or a bad thing? From a configuration management perspective it is definitely a bad thing, especially if the database being compared is the production database. I can speak from personal experience that making any changes to production databases, no matter how well-intentioned or trivial seeming, can lead to disaster and ruin.

No one even considers going in to a production application's code and "tweaking" a few things to improve performance. The code is changed in source control, built, and deployed to a test environment to ensure that the changes accomplished their desired effect. Then the changes are deployed to production. However, ad-hoc changes to the database are commonplace, seen as accepted practice when the truth is that they should not be tolerated. Adding the index may improve performance dramatically, but the index may also hurt performance, not to mention that adding the index straight to the database introduces the human element to the picture where a mistyped command can lead to the horrible sound of a database crashing and a résumé being prepared.

The safe course is the one where the database is treated equally with application code. The index is added to the database project, then built and tested just like any other change. Once the testing confirms that the index indeed improves application performance it can be packaged for deployment. The same approach is appropriate for all database schema changes, regardless of how trivial they may seem. It is imperative that we always know where the truth is, and the truth should be reflected in source control.

The second tool for analyzing database change is the data compare tool.


New Data Comparison

Choose Source and Target Databases

Source Database: Praxis.TailspinToys (PRAXIS\Dan)

Target Database: Praxis.TailspinToysDev (PRAXIS\Dan)

Data Compare Options

Records to compare:

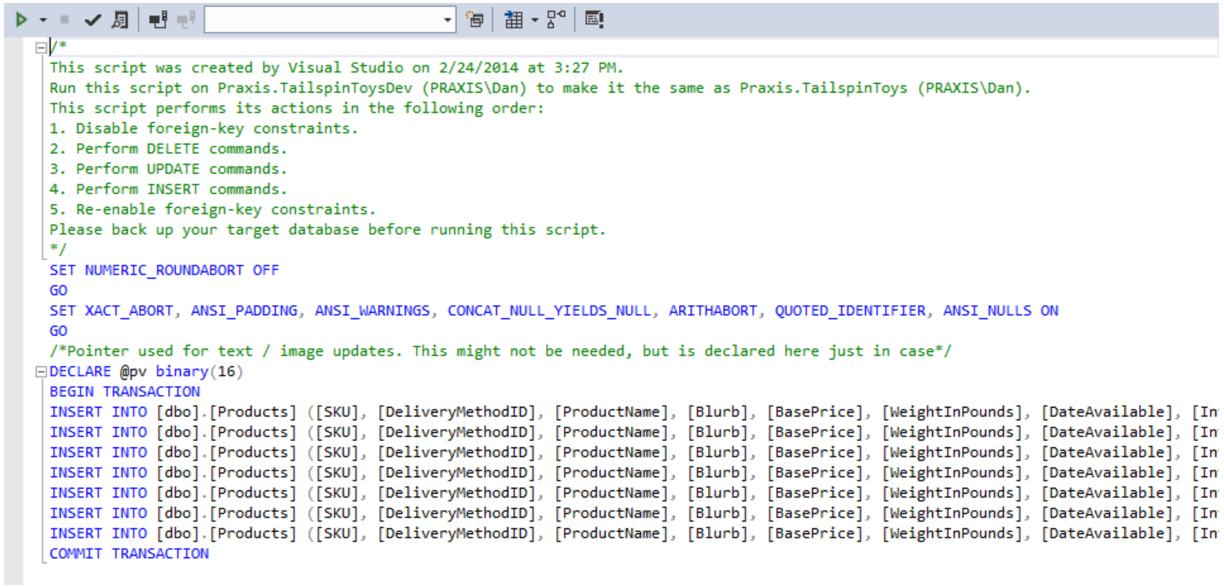
- [✓] Different Records
- [✓] Only in Source
- [✓] Only in Target
- [✓] Identical Records

Compare target database to source database with the option to synchronize the target database to the source database.

Data Comparison Tool

The data comparison tool can be configured to compare tables and/or views, and the comparison is limited to specific tables or views. This way, the comparison can look for changes made to lookup tables and tables containing metadata, and ignore user generated data.

The data comparison tool is very useful in finding rogue or erroneous data that impacts the application, but I personally think its best use is creating scripts that deploy database data changes, since once the comparison is complete the tool can output a script to apply the data changes. For example, using the database project, we can create a brand new database. However, the database is empty. It contains no metadata for the application, no product catalog- no information at all. It is essentially useless. However, running the data comparison tool and comparing a live database with the one created from source control can create a script that inserts all the required data for the application to function.



```
/*
This script was created by Visual Studio on 2/24/2014 at 3:27 PM.
Run this script on Praxis.TailspinToysDev (PRAXIS\Dan) to make it the same as Praxis.TailspinToys (PRAXIS\Dan).
This script performs its actions in the following order:
1. Disable foreign-key constraints.
2. Perform DELETE commands.
3. Perform UPDATE commands.
4. Perform INSERT commands.
5. Re-enable foreign-key constraints.
Please back up your target database before running this script.
*/
SET NUMERIC_ROUNDABORT OFF
GO
SET XACT_ABORT, ANSI_PADDING, ANSI_WARNINGS, CONCAT_NULL_YIELDS_NULL, ARITHABORT, QUOTED_IDENTIFIER, ANSI_NULLS ON
GO
/*Pointer used for text / image updates. This might not be needed, but is declared here just in case*/
DECLARE @pv binary(16)
BEGIN TRANSACTION
INSERT INTO [dbo].[Products] ([SKU], [DeliveryMethodID], [ProductName], [Blurb], [BasePrice], [WeightInPounds], [DateAvailable], [In
INSERT INTO [dbo].[Products] ([SKU], [DeliveryMethodID], [ProductName], [Blurb], [BasePrice], [WeightInPounds], [DateAvailable], [In
INSERT INTO [dbo].[Products] ([SKU], [DeliveryMethodID], [ProductName], [Blurb], [BasePrice], [WeightInPounds], [DateAvailable], [In
INSERT INTO [dbo].[Products] ([SKU], [DeliveryMethodID], [ProductName], [Blurb], [BasePrice], [WeightInPounds], [DateAvailable], [In
INSERT INTO [dbo].[Products] ([SKU], [DeliveryMethodID], [ProductName], [Blurb], [BasePrice], [WeightInPounds], [DateAvailable], [In
INSERT INTO [dbo].[Products] ([SKU], [DeliveryMethodID], [ProductName], [Blurb], [BasePrice], [WeightInPounds], [DateAvailable], [In
INSERT INTO [dbo].[Products] ([SKU], [DeliveryMethodID], [ProductName], [Blurb], [BasePrice], [WeightInPounds], [DateAvailable], [In
COMMIT TRANSACTION
```

Data Comparison Script

Once the script is created, it can be checked in to source control and called as part of the build process. It can also be updated when new lookup data is added.

The database itself is of little use without the data inside it that our application depends on, such as product information, shipping instructions and the like; all that wonderful metadata that brings meaning to the application. The data comparison tool makes it easy to create the scripts that bring meaning to the data base.

For testing purposes, we may want at least a sample of realistic user data as well. In older versions of Visual Studio (2008 and 2010), adding data to the database was done with a built-in data generation tool which could either insert random type-correct data or realistic data from a known source. That functionality does not exist in Visual Studio 2013 and Microsoft has no plans on developing a data generation tool in the future. The solution to this need depends on a great deal of variables that I won't go in to at this time. Suffice it to say that the best solution is one that is repeatable and can be called from the build process. There are a number of third-party solutions available. If large amounts of user data are not needed, scripts generated with the data comparison tool will work great. For test environments where very large amounts of data are required, I would recommend some other solution. Keeping a test database populated with realistic data, such as cleansed data harvested from production, is a common practice for performance testing environments. Functional test environments can typically use a much smaller user data set. Populating test databases is the trickiest part of the database development puzzle and needs to be approached with sustainability and maintenance forefront in the planning.

In this session, I explained the way to bring the database in to the solution. It isn't difficult. What is difficult is maintaining the discipline. There are years of habits that need to be unlearned. I won't say bad habits, because for many years, there were very few realistic alternatives to what I call the "Cowboy" approach to database development and modification. That was then. Today there are many alternatives and they are all better than letting the database go its own way, because like a sheep, it will go astray if left to itself. Do yourself a favor- Gather in the sheep and pen them up in source control.

The next article will focus on packaging the database and application code together for a single deployment. I will discuss the fascinating topic of rollbacks and database deployments using Visual Studio paired with a couple of my favorite third-party tools. Stay tuned!

About Dan Wood

Dan Wood is an ALM Consultant with Northwest Cadence. He is a huge rugby fan with an extensive background in database development and architecture and is a passionate advocate of agile methodologies. Dan has more than 20 years' experience in information technology and has extensive experience as a senior database architect, developer and administrator. He has authored and co-authored several books on SQL Server and the SQL language. In addition to designing and implementing many successful TFS installations, he has consulted and trained organizations of all sizes on end-to-end ALM with Visual Studio. Dan spent 20 years on submarines in the U.S. Navy and has a passion for getting it right the first time.

Partner Overview

Northwest Cadence partners with software development teams to improve processes and business results.

Recognized by Microsoft as the first Gold ALM Partner in the United States focusing exclusively on Software Development Process and Application Lifecycle Management, Northwest Cadence has provided professional services dating back to Microsoft ALM product inception with Visual Studio Team System in 2005. This experience coupled with the niche focus ensures clients benefit from consistently deep technical expertise, sound process acumen, and best practices learned through hundreds of practical experiences on agile transformation and ALM projects.

Contact Information – Phone 425.455.1155 | Website www.nwcadence.com | Blog blog.nwcadence.com

Northwest Cadence Corporation. All Rights Reserved. www.nwcadence.com | 425.455.1155