Getting started with ReadyRoll

This page helps you get started with developing and deploying databases with ReadyRoll. It takes you through different methods of working with migrations in ReadyRoll database projects.

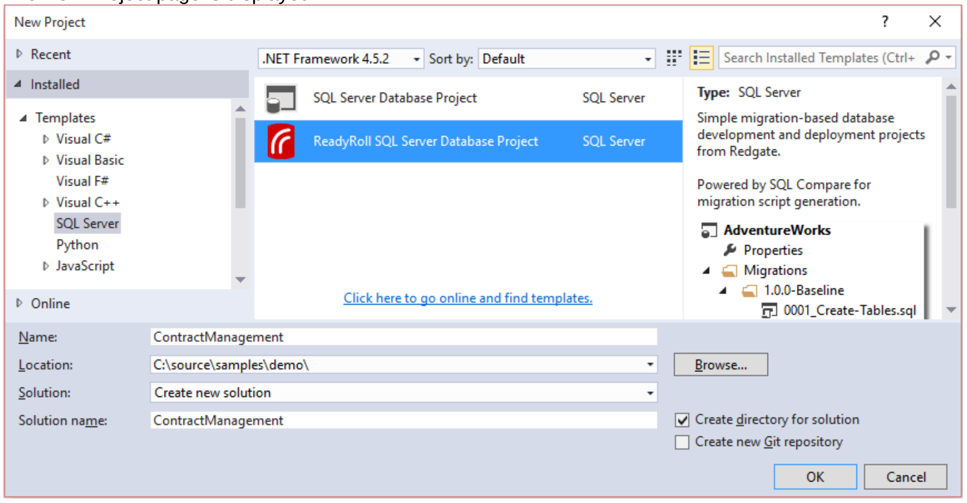
- Method 1: create new objects by using the Visual Studio designers to generate migration scripts in the project.
  This is a useful method when you're refactoring objects, because changes are captured in the generated script. The trade-off is that you may end up with many scripts in your project, increasing your team's maintenance and code-review overheads.
- Method 2: make online changes to your database, and import the new/modified object(s) into the project when ready.
  This is a useful method if you want to make multiple changes to your database without scripting a migration for each change. The trade-off is that, given the process used to generate scripts is limited to comparing the before-and-after state of the schema, some fidelity in the changes may be lost. For example, if you add a nullable column to a table, update it with data and then make the column NOT NULL, the generated script will fail to execute as it will lack the intermediate step of adding the nullable column (in such cases, an import should be performed after performing the first step).

In this tutorial we'll:

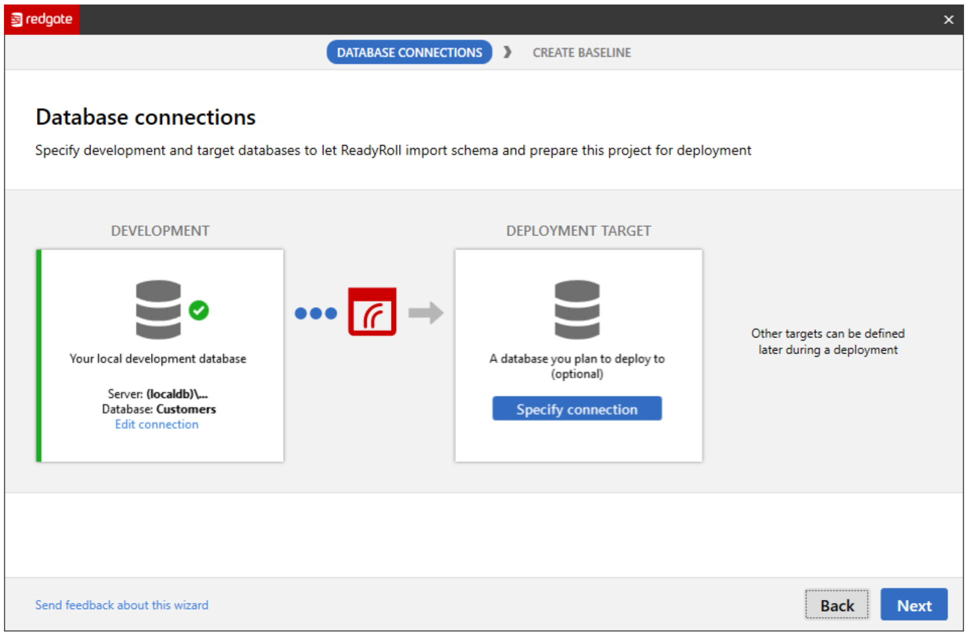
- Create a ReadyRoll database project in Visual Studio
- Design a table and generate your first migration (method 1)
- Create a view by editing the database online, and post-scripting a migration (method 2)

If you want to get started with an existing database, see Deploying to an existing database.

Create a ReadyRoll database project

1. In Visual Studio, from the **File** menu, select **New > Project**.
   The New Project page is displayed:
   

   Type: SQL Server
   Simple migration-based database development and deployment projects from Redgate.
   Powered by SQL Compare for migration script generation.
   Name: ContractManagement
   Location: C:\source\samples\demo\
   Solution: Create new solution
   Solution name: ContractManagement
   [✓] Create directory for solution
   [ ] Create new Git repository
2. From the **Installed > Templates > SQL Server** menu, select **ReadyRoll SQL Server Database Project**.
3. Enter the project name *ContractManagement*.
   This will also be the default name of your database.
4. You will be presented with a setup wizard. Follow the instructions to set your development database.


redgote

DATABASE CONNECTIONS › CREATE BASELINE

Database connections

Specify development and target databases to let ReadyRoll import schema and prepare this project for deployment

DEVELOPMENT

Your local development database

Server: (localdb)\...
Database: Customers
Edit connection

DEPLOYMENT TARGET

A database you plan to deploy to (optional)

Specify connection

Other targets can be defined later during a deployment

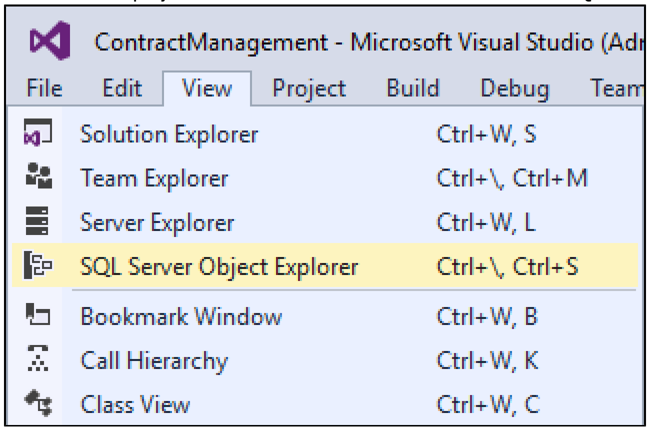
Send feedback about this wizard

Back Next

Note

You only need to specify a database in the area marked **Development**. Do not specify anything under the area marked as **Deployment Target**. This is used for specifying an existing database you would like to create a baseline schema from. For more information see Deploying to an existing database.

5. Click **OK**.
6. Once the new project is initialized, from the **View menu**, select **SQL Server Object Explorer**.


ContractManagement - Microsoft Visual Studio (Adr

File Edit View Project Build Debug Team

Solution Explorer	Ctrl+W, S
Team Explorer	Ctrl+\, Ctrl+M
Server Explorer	Ctrl+W, L
SQL Server Object Explorer	Ctrl+\, Ctrl+S
Bookmark Window	Ctrl+W, B
Call Hierarchy	Ctrl+W, K
Class View	Ctrl+W, C

This will allow you to view objects in the development database you specified in the wizard.

Design a table and generate your first migration

In this step, we'll use Visual Studio designers to deploy a table in the newly-created schema.

1. Within the SQL Server Object Explorer window, expand the **SQL Server** node and expand the appropriate SqlLocalDb instance for your version of Visual Studio.

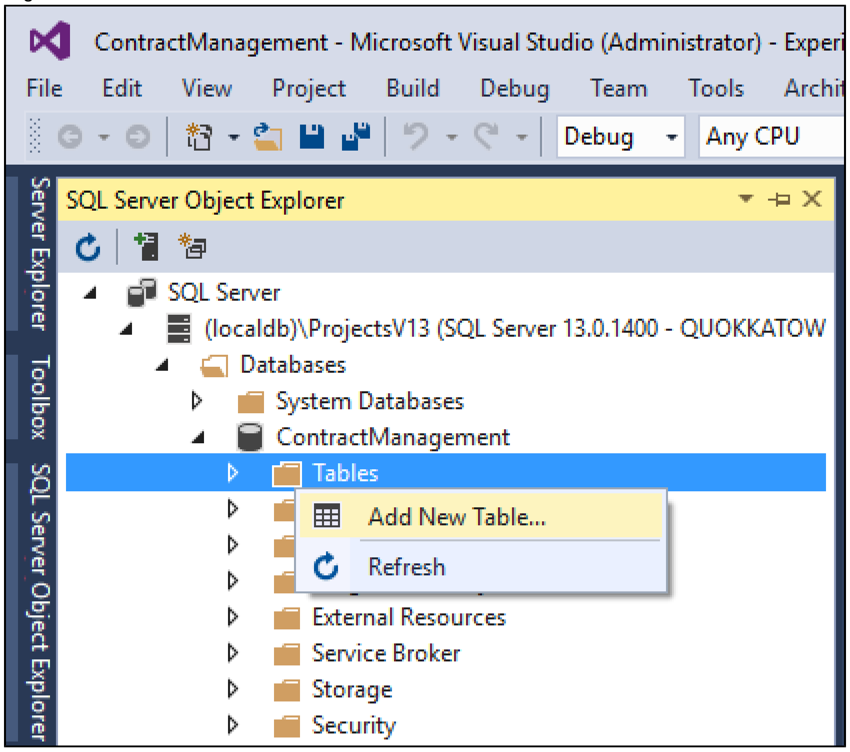
LocalDb Instance

An instance of SqlLocalDb is created automatically when you install Visual Studio. The name of the instance varies depending on which version of Visual Studio you have installed:

Visual Studio 2010: **(localdb)\Projects**
Visual Studio 2012. **(localdb)\Projects** or **(localdb)\ProjectsV11**
Visual Studio 2013: **(localdb)\Projects** or **(localdb)\ProjectsV12**
Visual Studio 2015 prior to Update 5: **(localdb)\ProjectsV12**
Visual Studio 2015 Update 5 onwards: **(localdb)\ProjectsV13**

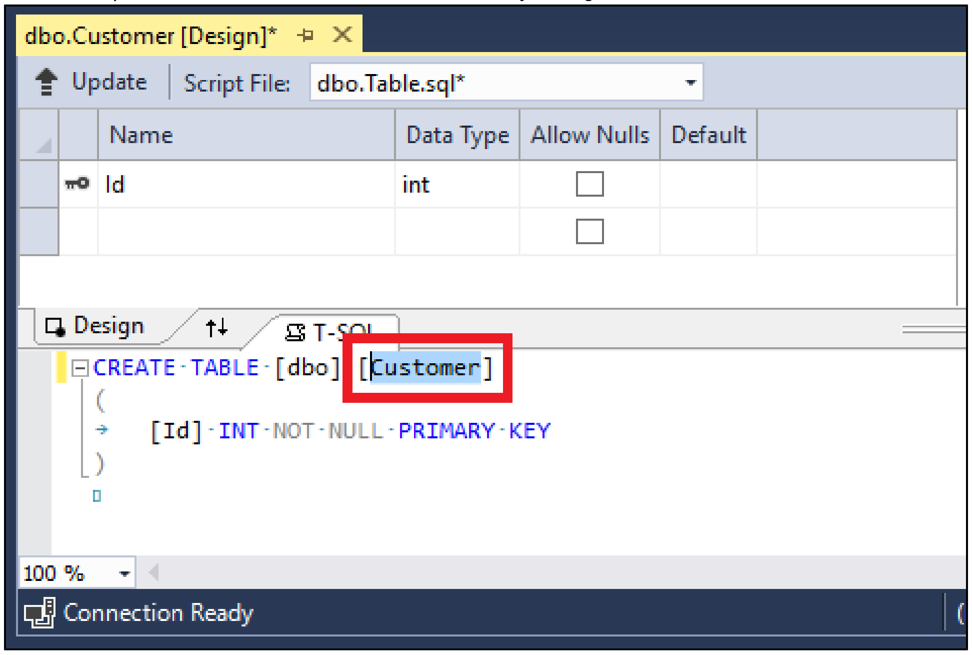
2. Expand the **Databases** node and then expand the *ContractManagement* database.

3. Right-click on **Tables** and select **Add New Table**.



The table designer is displayed. It contains two panes:
- an upper pane that you can use to add new columns, specify data types, default values, constraints, etc
- a lower pane that displays the object's declarative T-SQL, and lets you set the name of the table

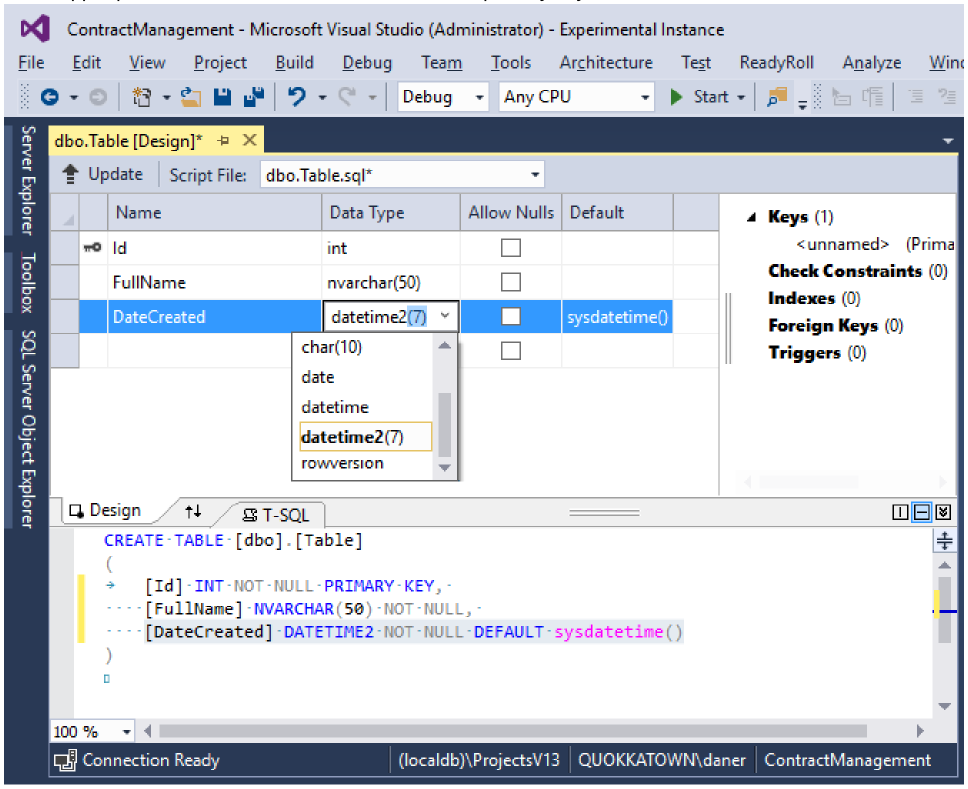
4. In the lower pane, set the name of the table to *Customer* by editing the CREATE TABLE statement:



Name	Data Type	Allow Nulls	Default
Id	int	☐	
		☐	

```
CREATE TABLE [dbo].[Customer]
(
    [Id] INT NOT NULL PRIMARY KEY
)
```

5. In the upper pane, add columns to the table and set a primary key, as follows:



ContractManagement - Microsoft Visual Studio (Administrator) - Experimental Instance

File Edit View Project Build Debug Team Tools Architecture Test ReadyRoll Analyze Wind

Debug | Any CPU | Start

dbo.Table [Design]*

Update | Script File: dbo.Table.sql*

Name	Data Type	Allow Nulls	Default
Id	int	☐	
FullName	nvarchar(50)	☐	
DateCreated	datetime2(7)	☐	sysdatetime()
		☐	

char(10)
date
datetime
datetime2(7)
rowversion

Keys (1)
<unnamed> (Prima
Check Constraints (0)
Indexes (0)
Foreign Keys (0)
Triggers (0)

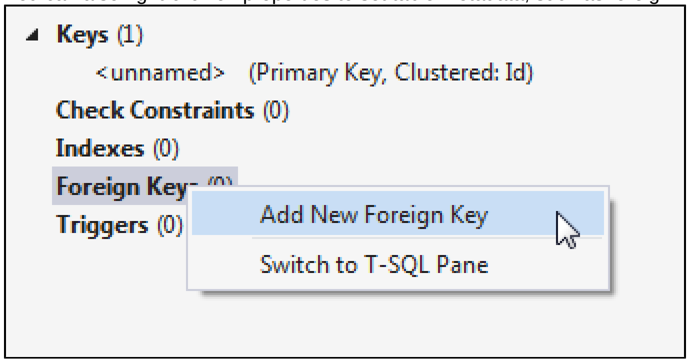
Design | T-SQL

```
CREATE TABLE [dbo].[Table]
(
    [Id] INT NOT NULL PRIMARY KEY,
    [FullName] NVARCHAR(50) NOT NULL,
    [DateCreated] DATETIME2 NOT NULL DEFAULT sysdatetime()
)
```

100 %

Connection Ready | (localdb)\ProjectsV13 | QUOKKATOWN\daner | ContractManagement

You can also right-click on properties to set table metadata, such as foreign keys, constraints and indexes:

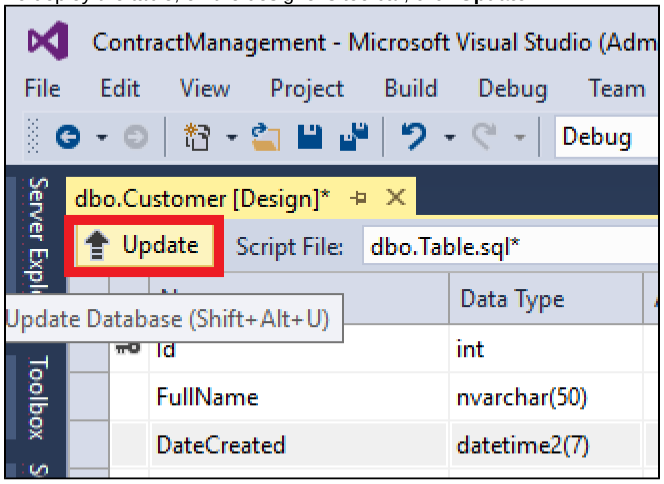


Keys (1)
<unnamed> (Primary Key, Clustered: Id)
Check Constraints (0)
Indexes (0)
Foreign Keys (0)
Triggers (0)

Even though this contains the DDL to deploy the object, the T-SQL will always display a *CREATE* statement, even if you modify the object after deployment.

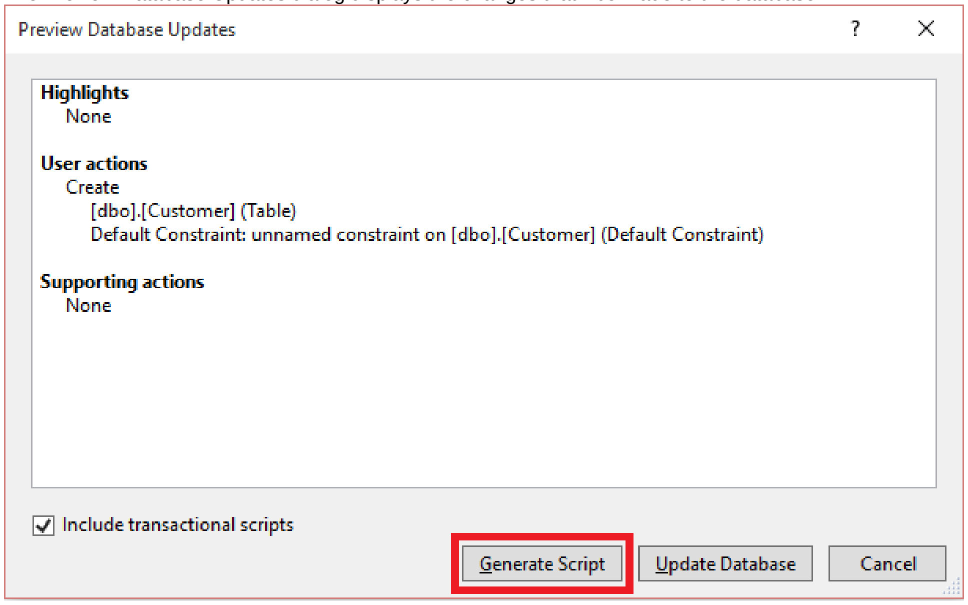
We'll cover how SSDT generates an *ALTER* statement to sync table modifications later in the tutorial.

6. To deploy the table, on the designer's toolbar, click **Update**.



Name	Data Type
Id	int
FullName	nvarchar(50)
DateCreated	datetime2(7)

The Preview Database Updates dialog displays the changes that'll be made to the database:

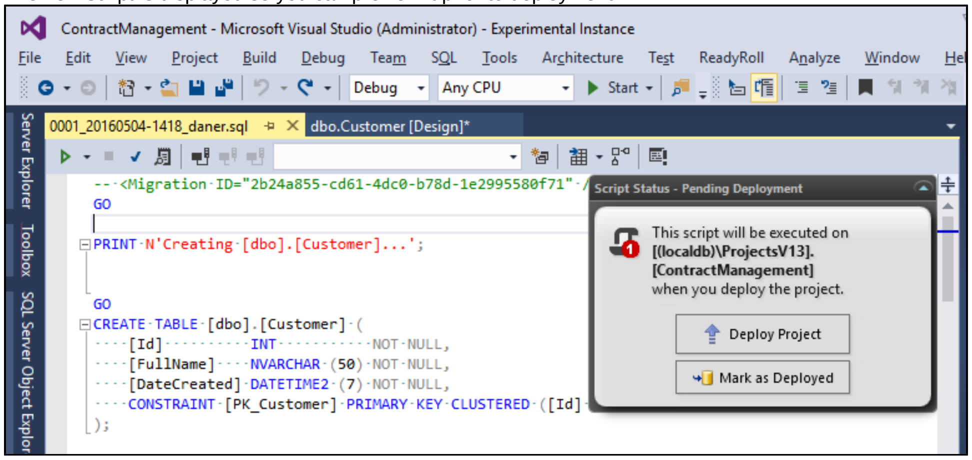


Highlights
None

User actions
Create
[dbo].[Customer] (Table)
Default Constraint: unnamed constraint on [dbo].[Customer] (Default Constraint)

Supporting actions
None

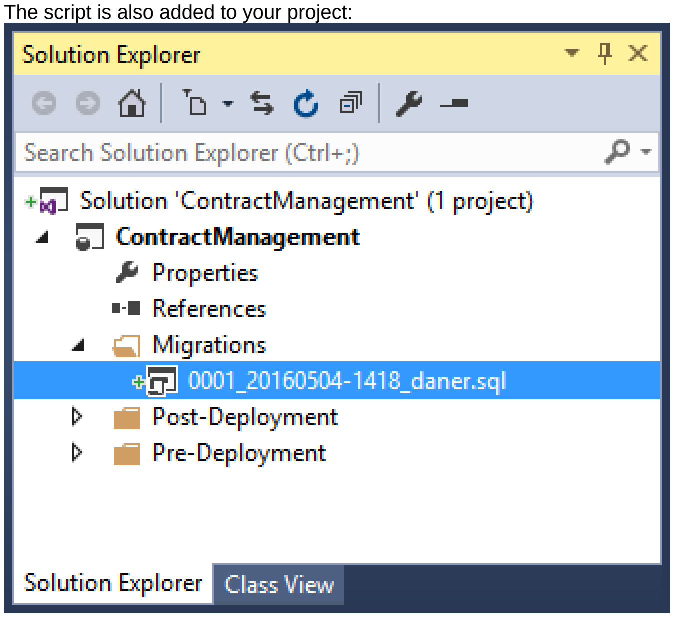
7. Click **Generate Script**.
   The new script is displayed so you can preview it prior to deployment:


ContractManagement - Microsoft Visual Studio (Administrator) - Experimental Instance
File Edit View Project Build Debug Team SQL Tools Architecture Test ReadyRoll Analyze Window
Debug
Any CPU
Start
0001_20160504-1418_daner.sql
dbo.Customer [Design]*
Server Explorer
Toolbox
SQL Server Object Explor
Script Status - Pending Deployment
Deploy Project
Mark as Deployed

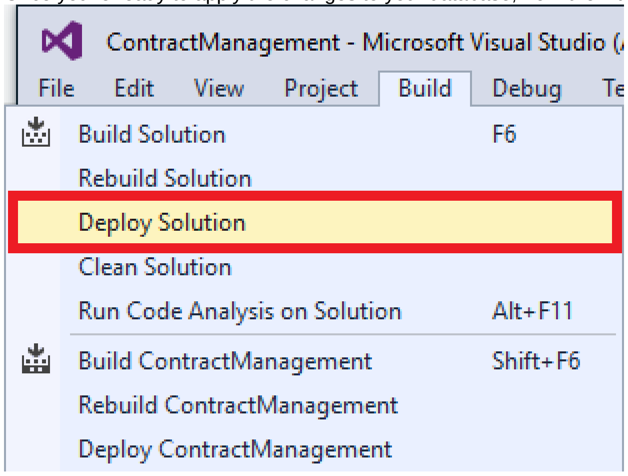
```
-- <Migration ID="2b24a855-cd61-4dc0-b78d-1e2995580f71" /
GO

PRINT N'Creating [dbo].[Customer]...';


GO
CREATE TABLE [dbo].[Customer] (
    [Id]          INT           NOT NULL,
    [FullName]    NVARCHAR (50) NOT NULL,
    [DateCreated] DATETIME2 (7) NOT NULL,
    CONSTRAINT [PK_Customer] PRIMARY KEY CLUSTERED ([Id]
);
```

This script will be executed on **[(localdb)\ProjectsV13].[ContractManagement]** when you deploy the project.

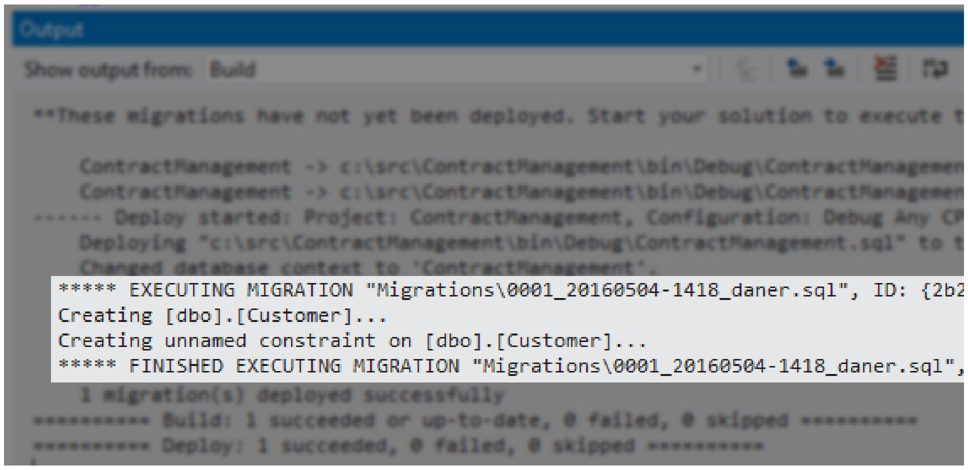
   The script is also added to your project:



8. Once you're ready to apply the changes to your database, from the **Build** menu, select **Deploy Solution**:

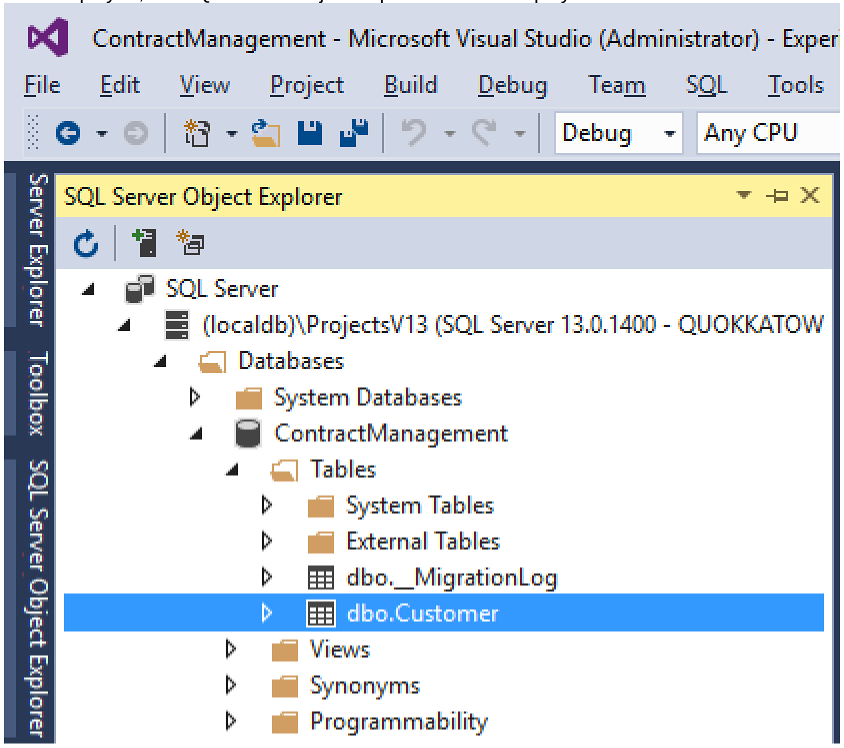

ContractManagement - Microsoft Visual Studio (
File Edit View Project Build Debug Te
Build Solution F6
Rebuild Solution
Deploy Solution
Clean Solution
Run Code Analysis on Solution Alt+F11
Build ContractManagement Shift+F6
Rebuild ContractManagement
Deploy ContractManagement

   View the deployment results in the Output window:

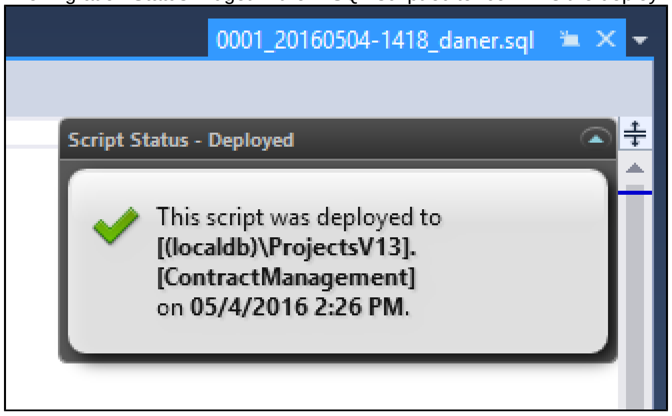


```
**These migrations have not yet been deployed. Start your solution to execute t

    ContractManagement -> c:\src\ContractManagement\bin\Debug\ContractManagemen
    ContractManagement -> c:\src\ContractManagement\bin\Debug\ContractManagemen
------ Deploy started: Project: ContractManagement, Configuration: Debug Any CP
    Deploying "c:\src\ContractManagement\bin\Debug\ContractManagement.sql" to t
    Changed database context to 'ContractManagement'.
***** EXECUTING MIGRATION "Migrations\0001_20160504-1418_daner.sql", ID: {2b2
Creating [dbo].[Customer]...
Creating unnamed constraint on [dbo].[Customer]...
***** FINISHED EXECUTING MIGRATION "Migrations\0001_20160504-1418_daner.sql",
    1 migration(s) deployed successfully
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
========== Deploy: 1 succeeded, 0 failed, 0 skipped ==========
```

Once deployed, the SQL Server Object Explorer window displays the new table in the database schema:



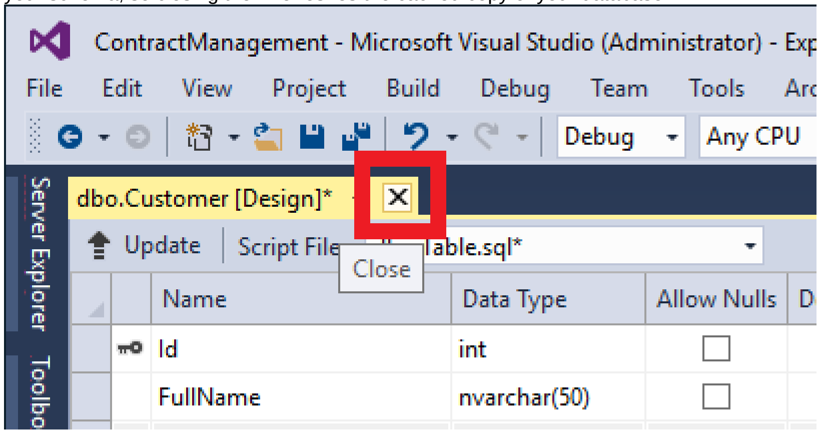
The Migration Status widget in the T-SQL script editor confirms the deployment:



This script was deployed to **[(localdb)\ProjectsV13].[ContractManagement]** on **05/4/2016 2:26 PM.**

You can also deploy by starting the solution [F5]. However, if you happen to have a different project set as the Start-up Project in your solution, you'll need to add a reference from that project to the database project to make sure that the database is also built and deployed.

Once your object has been successfully deployed, close any open designers.The designers in *SQL Server Data Tools* keep an in-memory copy of your schema, so closing them refreshes the cached copy of your database.



If you're prompted to save changes, click **No**. Changes have been applied already.

You've now created new objects by deploying changes via the project as migration scripts. If you want to modify existing objects, the process is the same.

Create a view using the online-editing method

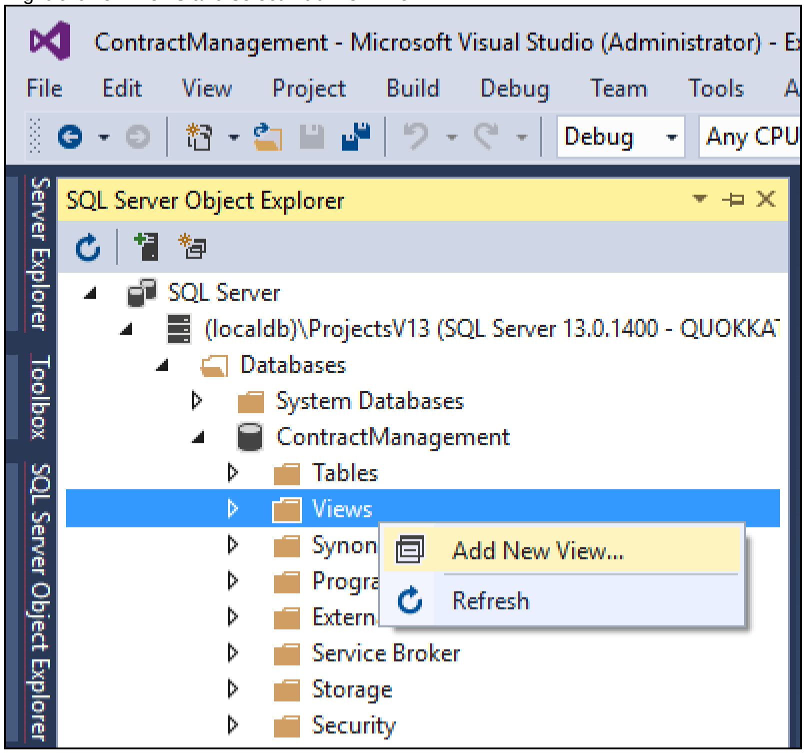
Instead of scripting a database object before deployment, we're now going to save changes directly in the database, and then create a migration script by importing the changes into the project at the end.

To add a view to the database:

1. Within the SQL Server Object Explorer window, expand the **SQL Server** node and expand the appropriate SqlLocalDb instance for your version of Visual Studio (see the list above).

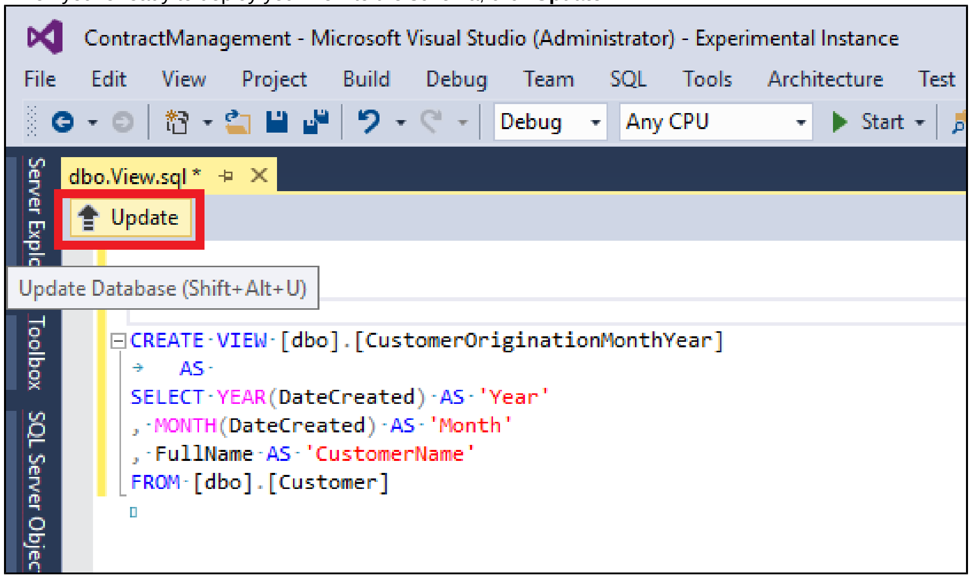
2. Expand the **Databases** node and then expand the *ContractManagement* database node.

3. Right-click on **Views** and select **Add New View**.


ContractManagement - Microsoft Visual Studio (Administrator) - E
File Edit View Project Build Debug Team Tools A
Debug
Any CPU
Server Explorer
Toolbox
SQL Server Object Explorer
SQL Server Object Explorer
SQL Server
(localdb)\ProjectsV13 (SQL Server 13.0.1400 - QUOKKA
Databases
System Databases
ContractManagement
Tables
Views
Synon
Progra
Extern
Service Broker
Storage
Security
Add New View...
Refresh

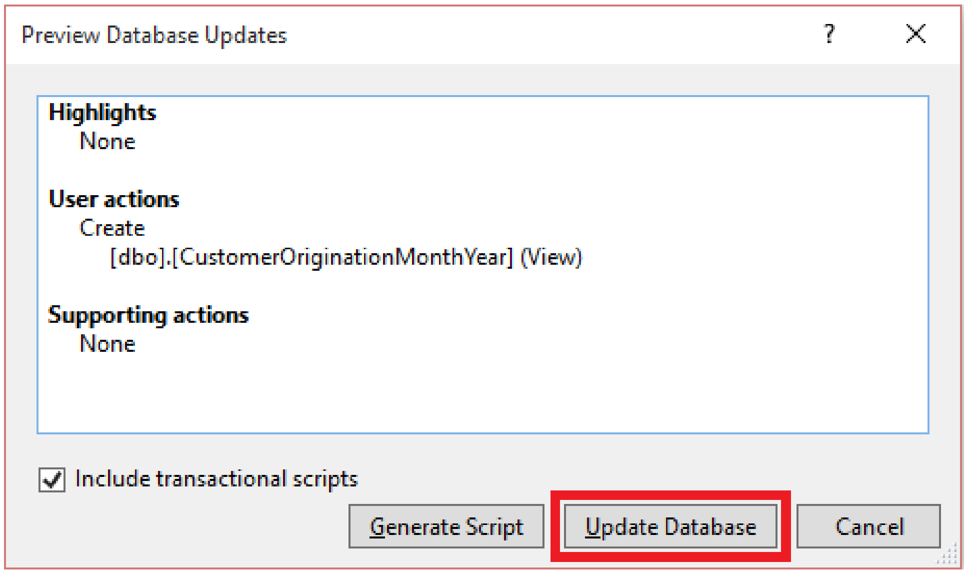
4. Enter a query to create the view. The code editor provides IntelliSense to help you pick column names.

5. When you're ready to deploy your view to the schema, click **Update**.


ContractManagement - Microsoft Visual Studio (Administrator) - Experimental Instance
File Edit View Project Build Debug Team SQL Tools Architecture Test
Debug
Any CPU
Start
dbo.View.sql *
Update
Update Database (Shift+Alt+U)
Server Explo
Toolbox
SQL Server Objec

```
CREATE VIEW [dbo].[CustomerOriginationMonthYear]
    AS
SELECT YEAR(DateCreated) AS 'Year'
, MONTH(DateCreated) AS 'Month'
, FullName AS 'CustomerName'
FROM [dbo].[Customer]
```

6. Click **Update Database** to deploy your object to the schema:



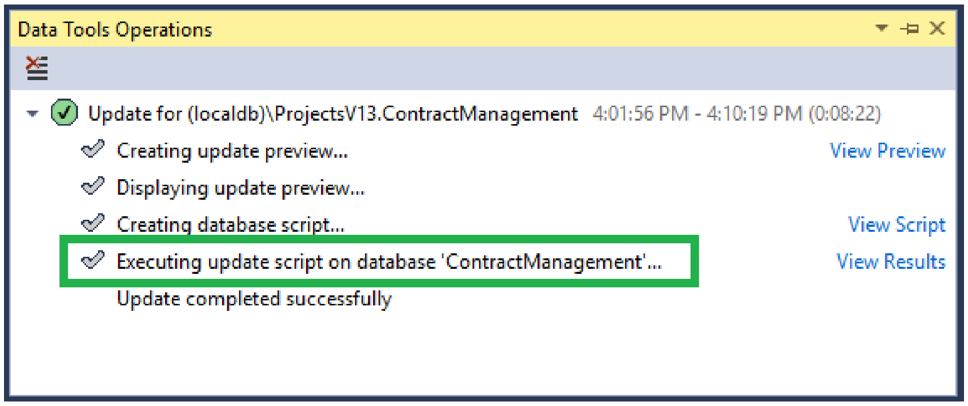
Highlights
None

User actions
Create
[dbo].[CustomerOriginationMonthYear] (View)

Supporting actions
None

[✓] Include transactional scripts

Changes are applied directly to the target database, skipping the 'generate script' step. You can make additional edits if necessary, before you generate a delta script of pending changes.

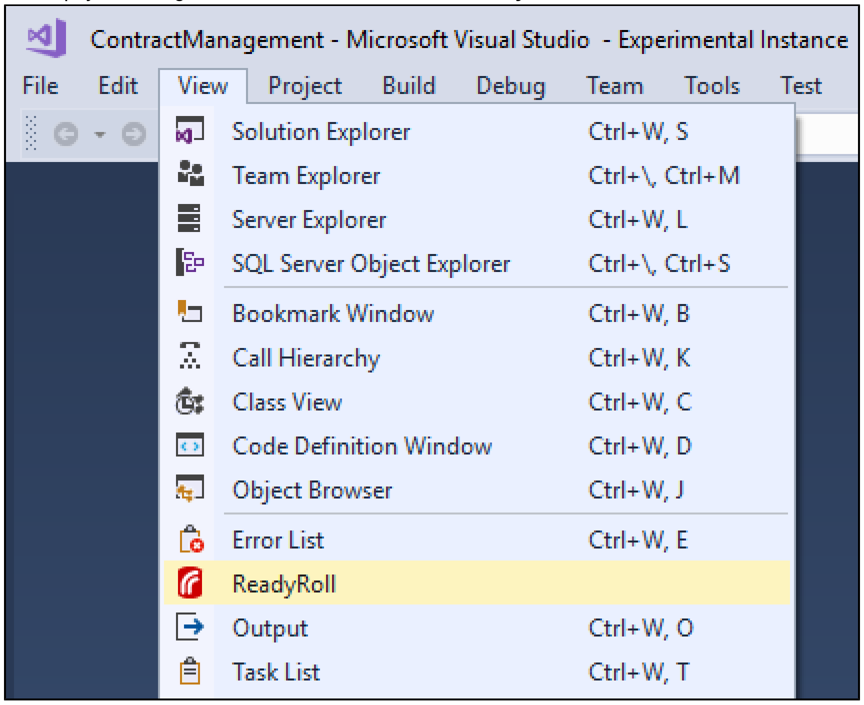


```
Update for (localdb)\ProjectsV13.ContractManagement   4:01:56 PM - 4:10:19 PM (0:08:22)
    Creating update preview...
    Displaying update preview...
    Creating database script...
    Executing update script on database 'ContractManagement'...
    Update completed successfully
```

If you prefer, you can use SQL Management Studio (instead of Visual Studio) to make online changes to your database.

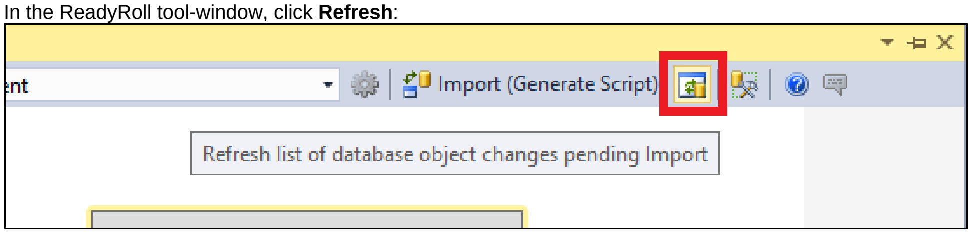
Once you've made changes in SSMS, switch back to Visual Studio and use the ReadyRoll tool-window to import them.

7. To script your changes, from the **View** menu, select **ReadyRoll**:


ContractManagement - Microsoft Visual Studio - Experimental Instance
File
Edit
View
Project
Build
Debug
Team
Tools
Test
Solution Explorer Ctrl+W, S
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+W, L
SQL Server Object Explorer Ctrl+\, Ctrl+S
Bookmark Window Ctrl+W, B
Call Hierarchy Ctrl+W, K
Class View Ctrl+W, C
Code Definition Window Ctrl+W, D
Object Browser Ctrl+W, J
Error List Ctrl+W, E
ReadyRoll
Output Ctrl+W, O
Task List Ctrl+W, T

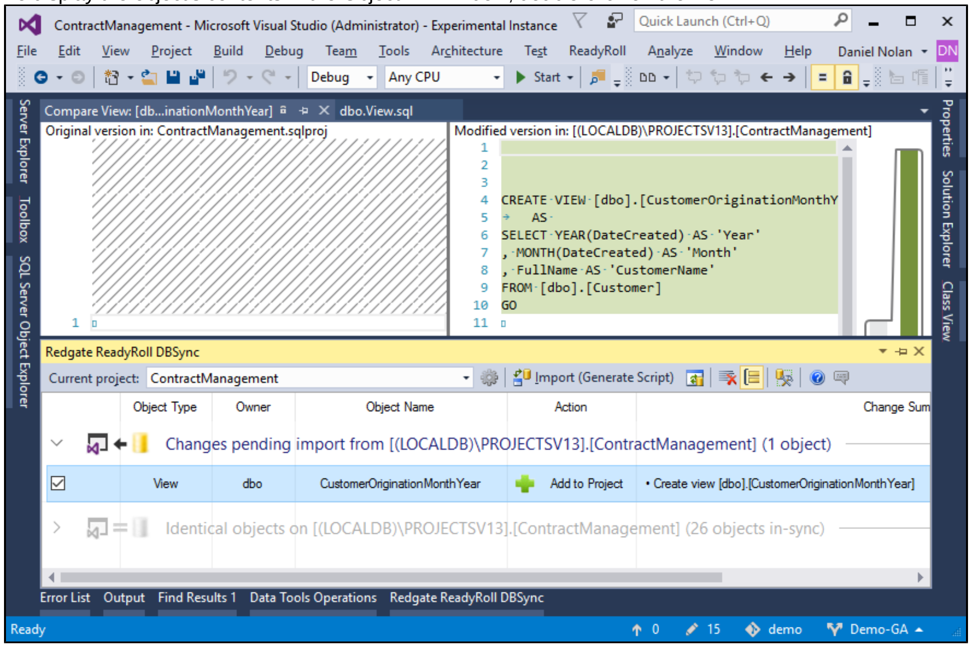
The ReadyRoll tool-window is displayed. You can use it to import schema objects from your sandbox (i.e. target) database, and revert changes to a previous known "good" state.

8. In the ReadyRoll tool-window, click **Refresh**:



ReadyRoll starts to compare the sandbox database to your project. The list of pending objects is displayed.

9. To display the object's contents in the Object Diff window, double-click on the view:



Original version in: ContractManagement.sqlproj

Modified version in: [(LOCALDB)\PROJECTSV13].[ContractManagement]

```



CREATE VIEW [dbo].[CustomerOriginationMonthY
	AS
SELECT YEAR(DateCreated) AS 'Year'
, MONTH(DateCreated) AS 'Month'
, FullName AS 'CustomerName'
FROM [dbo].[Customer]
GO

```

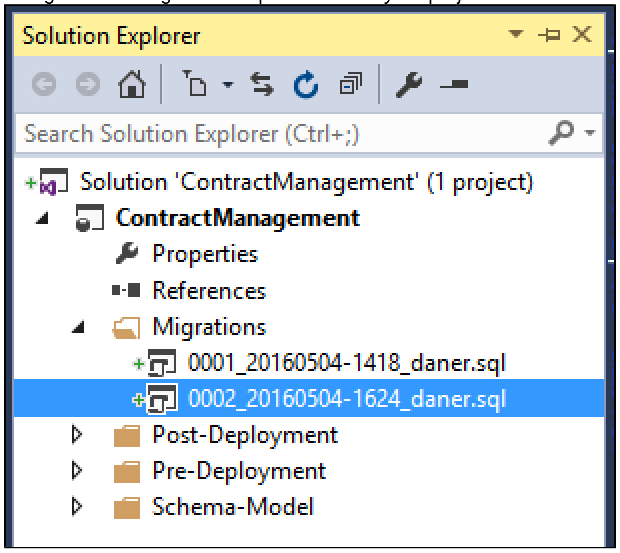
Redgate ReadyRoll DBSync

Current project: ContractManagement | Import (Generate Script)

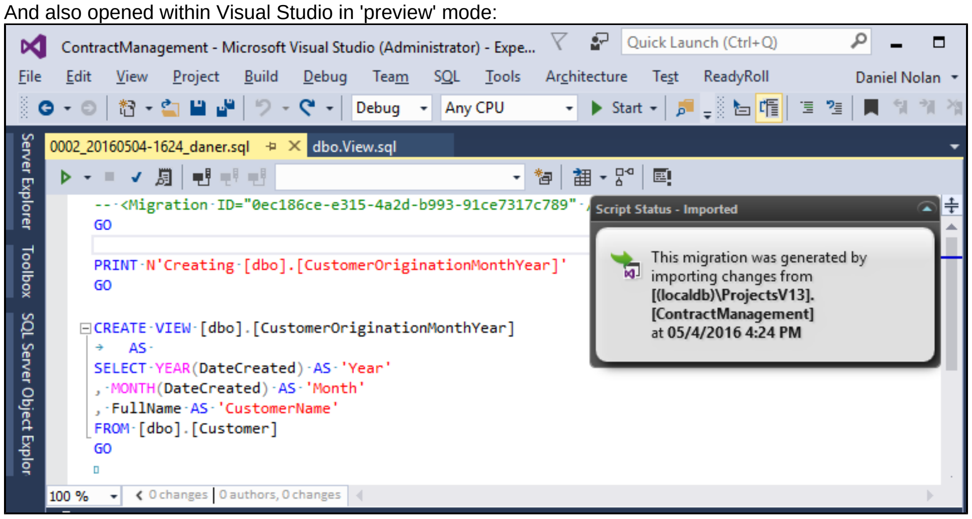
	Object Type	Owner	Object Name	Action	Change Sum
Changes pending import from [(LOCALDB)\PROJECTSV13].[ContractManagement] (1 object)					
✓	View	dbo	CustomerOriginationMonthYear	Add to Project	• Create view [dbo].[CustomerOriginationMonthYear]
Identical objects on [(LOCALDB)\PROJECTSV13].[ContractManagement] (26 objects in-sync)					

10. To import the changes into the project, click **Import (Generate Script)**.

The generated migration script is added to your project.


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ContractManagement' (1 project)
ContractManagement
Properties
References
Migrations
0001_20160504-1418_daner.sql
0002_20160504-1624_daner.sql
Post-Deployment
Pre-Deployment
Schema-Model

And also opened within Visual Studio in 'preview' mode:


ContractManagement - Microsoft Visual Studio (Administrator) - Expe...
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team SQL Tools Architecture Test ReadyRoll
Daniel Nolan
Debug
Any CPU
Start
0002_20160504-1624_daner.sql
dbo.View.sql
Server Explorer
Toolbox
SQL Server Object Explor
Script Status - Imported
100 %
0 changes
0 authors, 0 changes

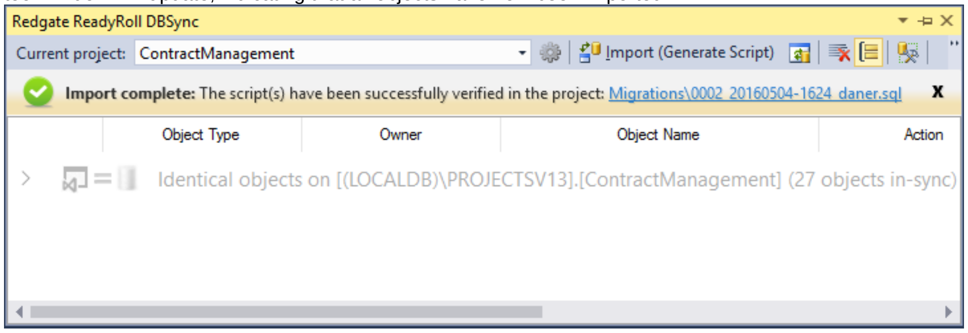
```
-- <Migration ID="0ec186ce-e315-4a2d-b993-91ce7317c789" /
GO

PRINT N'Creating [dbo].[CustomerOriginationMonthYear]'
GO

CREATE VIEW [dbo].[CustomerOriginationMonthYear]
    AS
SELECT YEAR(DateCreated) AS 'Year'
, MONTH(DateCreated) AS 'Month'
, FullName AS 'CustomerName'
FROM [dbo].[Customer]
GO
```

This migration was generated by importing changes from **[(localdb)\ProjectsV13].[ContractManagement]** at **05/4/2016 4:24 PM**

The script isn't executed against the sandbox database: its purpose is to allow your changes to be deployed to another server. I.e., it would only be executed on your sandbox if you were to drop and redeploy your sandbox database. For more information about the target database, see Target and shadow databases.

ⓘ Although any edits to the script will not be applied to your Sandbox database, you can make changes to the T-SQL if the generated script doesn't do what you expected. For example, if you rename a Table, ReadyRoll will actually script a `DROP/CREATE` instead of an `EXEC sp_rename`.

11. To test the script, click **Refresh (Verify Script)** within the tool-window.
The script is executed against the *Shadow* database in order to test the integrity of the migration. If the verification of your script is successful, the tool-window will update, indicating that all objects have now been imported.



Import complete: The script(s) have been successfully verified in the project: Migrations\0002_20160504-1624_daner.sql

Object Type	Owner	Object Name	Action
Identical objects on [(LOCALDB)\PROJECTSV13].[ContractManagement] (27 objects in-sync)			

12. You've now made online changes to your database, and imported the new/modified object(s) into the project.

Summary

You've now tried two different ways of making database changes and creating migration scripts in your ReadyRoll database projects.
Depending on how you like to work, you may choose one approach over the other. You may even combine the two methods: edit online when you want to create new tables or modify programmable objects, and generate migrations directly when you want to refactor table objects.