Second Edition

Developing with SQL Server®

Learn How to Build Robust Data Processes and Applications with SQL Server® and Related Technologies!

Artemakis Artemiou

Developing with SQL Server®

(Second Edition)

ARTEMAKIS ARTEMIOU

PUBLISHED BY
Artemakis Artemiou

Publication Date: October 11 2017.

About the Author

Artemakis Artemiou is a Senior SQL Server and Software Architect, Author, and a former Microsoft Data Platform MVP (2009-2018). He has over 15 years of experience in the IT industry in various roles. Artemakis is the founder of SQLNetHub and TechHowTos.com. Artemakis is the creator of the well-known software tools Snippets Generator, DBA Security Advisor and In-Memory OLTP Simulator. Moreover, he is the author of many eBooks on SQL Server. Artemakis currently serves as the President of the Cyprus .NET User Group (CDNUG) and the International .NET Association Country Leader for Cyprus (INETA). Artemakis's official website can be found at aartemiou.com. You can follow Artemakis on Twitter at https://twitter.com/artemakis.

Some of the online channels where you can find Artemakis are:

- SQLNetHub
- Official Website
- The SQL Server and .NET TV
- TechHowTos.com
- Twitter
- Cyprus .NET User Group

Contact the Author

You can contact the author using the contact methods on www.aartemiou.com and www.sqlnethub.com

To all of those who never give up chasing their dreams,
who are never satisfied with mediocrity in life,
who create opportunities from scratch,
who always try to be the best, with respect and support to their fellow human being,
who never stop exploring and never stop believing that anything is possible.
-A.A.

Introduction

I started working with SQL Server and .NET (C#) 15 years ago. Since then it has been quite a journey! Each release came - and still comes - with exciting new features enabling us to do more and more! Every time waiting for that new built, in order to start testing it, exploring it, learning it, as soon as it becomes available. The possibilities are endless! The only limit is your creativity!

The massive interaction with the SQL Server community started at about seven years ago. Blogging, organizing user group events, speaking in user group meetings, conferences and other events, open-source projects related to SQL Server, guest articles, discussions on message boards/forums, and much more!

The love for technical writing and knowledge sharing urged me for adding another activity to my interaction with the community that is ***book authoring***. In order to be able to write, you first need to acquire and comprehend the specific technical knowledge. You need to explore, to experiment, to test. You need to test the limits of each new technology or feature in order to be able to fully understand its nature and capabilities.

SQL Server and Azure SQL Database are very powerful data platforms that include several data management, analysis and data science technologies that allow you to do just about anything with your data. Since 2002 I have been exploring SQL Server in many areas. I have been deep diving into various topics of SQL server having to do mainly with: ***administration***, ***development/data access***, ***performance tuning, In-Memory OLTP***. These are the four areas of SQL Server I mostly focus on and on which I base, but not limit, my interaction with SQL Server and acquisition of knowledge. To this end, I have decided to start writing eBooks mainly, but not limited to, on the above areas.

Who Should Read This Book?

This book is for database developers and architects who develop SQL Server databases and related database processes. The book features tens of articles that provide detailed information on how to develop in SQL Server but also in Visual Studio. However, the book is not intended to be a step-by-step comprehensive guide. Additionally, it assumes at least intermediate-level experience with SQL Server development and knowledge of basic database principles.

Supported SQL Server Versions

Each article in this book has in the end an "***Applies to:***" note indicating the SQL Server versions against which the provided T-SQL scripts can be executed. There are also articles providing different T-SQL scripts for SQL Server 2000, SQL Server 2005 or later. Every time I write an article, I try to write code that

can be executed in all versions of SQL Server unless stated otherwise. Even though SQL Server 2000 and 2005 are not officially supported anymore, I am quite sure that still there are many of you out there who still use these versions of SQL Server for various reasons (i.e. application compatibility issues, etc.). My advice is to consider upgrading to a newer version of SQL Server the soonest possible, in order to be able to use the benefits of the latest SQL Server releases as well as being able to get official support and updates.

How Is This Book Organized?

This book is organized as follows. Chapter 1 discusses different programmability topics on SQL Server as well as ways to handle different problems you might encounter while developing database operations. Chapter 2 discusses how you can work with unstructured data in different versions of SQL Server. Chapter 3 discusses Data Access topics and more specifically the ADO.NET Entity Framework as well as WCF Data Services. Chapter 4 provides some straight to the point T-SQL tips for performing specific tasks. Chapter 5 features several special topics on SQL Server like where programmability objects are stored and how to retrieve information about them. Finally, Chapter 6 discusses miscellaneous topics on SQL Server and on generic database-related topics.

Feedback

I am a proud member of the worldwide SQL Server community. Feedback from you, my fellow community members, is more than welcome. Please feel free to visit the following link and provide your feedback:

http://www.sqlnethub.com/feedback/

The survey is short. It will only take 2 minutes of your valuable time.

Acknowledgments

Writing a book is not an easy thing. It doesn't really matter the length of the book. Even if you just write a few pages, it takes a considerable amount of time and energy. In the case where you are not a professional technical writer but just a technical community guy like me, this amount of time is valuable "free" time after work, time from your family and beloved ones. You may use this time because you are just passionate in what you do. However, this is not enough, at least to me. Without the support of your family it just doesn't feel right. During this process I had all the support I needed. I am very grateful for all the support my beautiful wife and daughters give me when writing books and articles and for all their support and love in everything I do. Without their support and encouragement none of this would have been possible. This book is dedicated to them with all my love.

- Artemakis

Stay in Touch

Feel free to connect! You can find me on the following online channels:

- SQLNetHub
- Official Website
- The SQL Server and .NET TV
- TechHowTos.com
- Twitter
- Cyprus .NET User Group

CHAPTER 2

Working with Unstructured Data

IN THIS CHAPTER:

- How to Import and Export Unstructured Data in SQL Server - The IMAGE Datatype
- How to Import and Export Unstructured Data in SQL Server – FILESTREAM
- How to Import and Export Unstructured Data in SQL Server – FileTables

Unstructured Data is a term that refers to information that does not have a predefined data model thus is not contained in a database or any other data structure. This type of data can be text, image files, audio and video files, etc.

This chapter features three articles that discuss ways of working with unstructured data in different versions of SQL Server thus utilizing the available features in each one of the SQL Server versions.

The first article discusses how you can import and export unstructured data in SQL Server 2005 using the IMAGE data type, the second article explains how you can do the same in SQL Server 2008 by using the FILESTREAM feature, and the third article discuses again how you can import and export unstructured data using the FileTables feature available in SQL Server 2012 or later.

How to Import and Export Unstructured Data in SQL Server - The IMAGE Datatype

Importing and exporting unstructured data from a database is a common practice, especially in cases of large applications that use multimedia. The FILESTREAM feature which was originally introduced in SQL Server 2008, allows storing unstructured data (i.e. music, video, documents, etc.) onto the NTFS file system and manipulating it via the Database Engine. Moreover, SQL Server 2012 introduced FileTables which is an enhancement to FILESTREAM.

In this article we are going to see how it is possible to import and export binary objects in SQL Server. Instead of saying more, I would just like to show you by example how you can do this.

For this example, consider that we have the following two objects:

- SampleImage.png
- SampleTextFile.txt

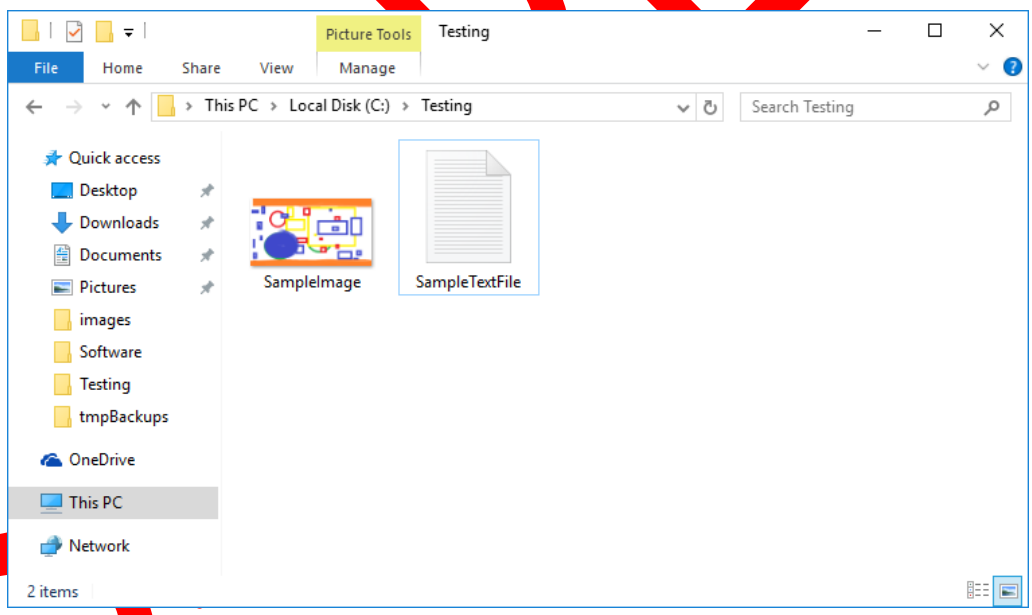

Figure 2.1: The two sample files for the BLOBs example.

Let's take a closer look at the files just by checking out their content:

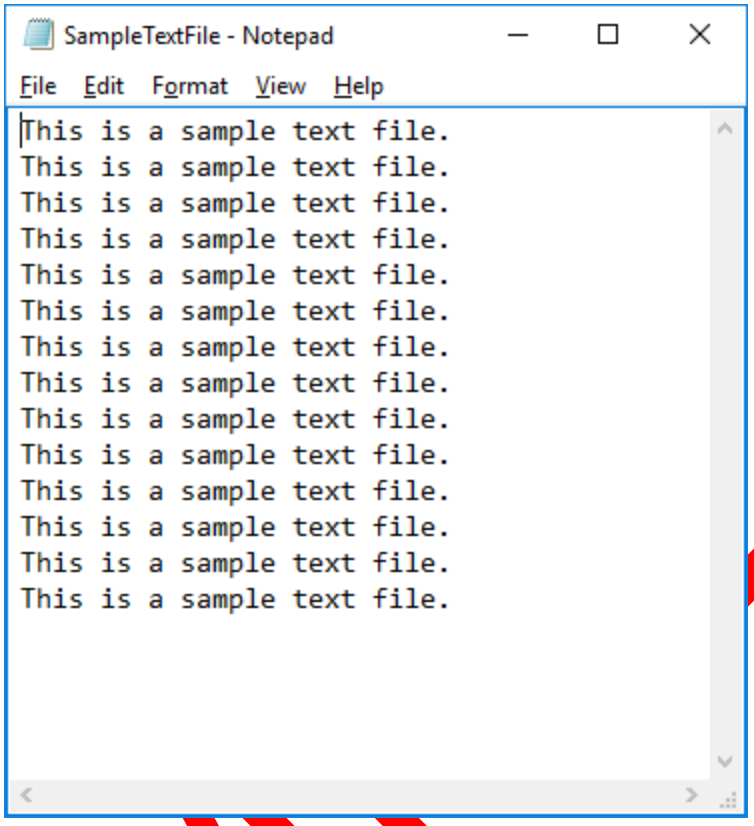

Figure 2.2: Content of SampleTextFile.txt

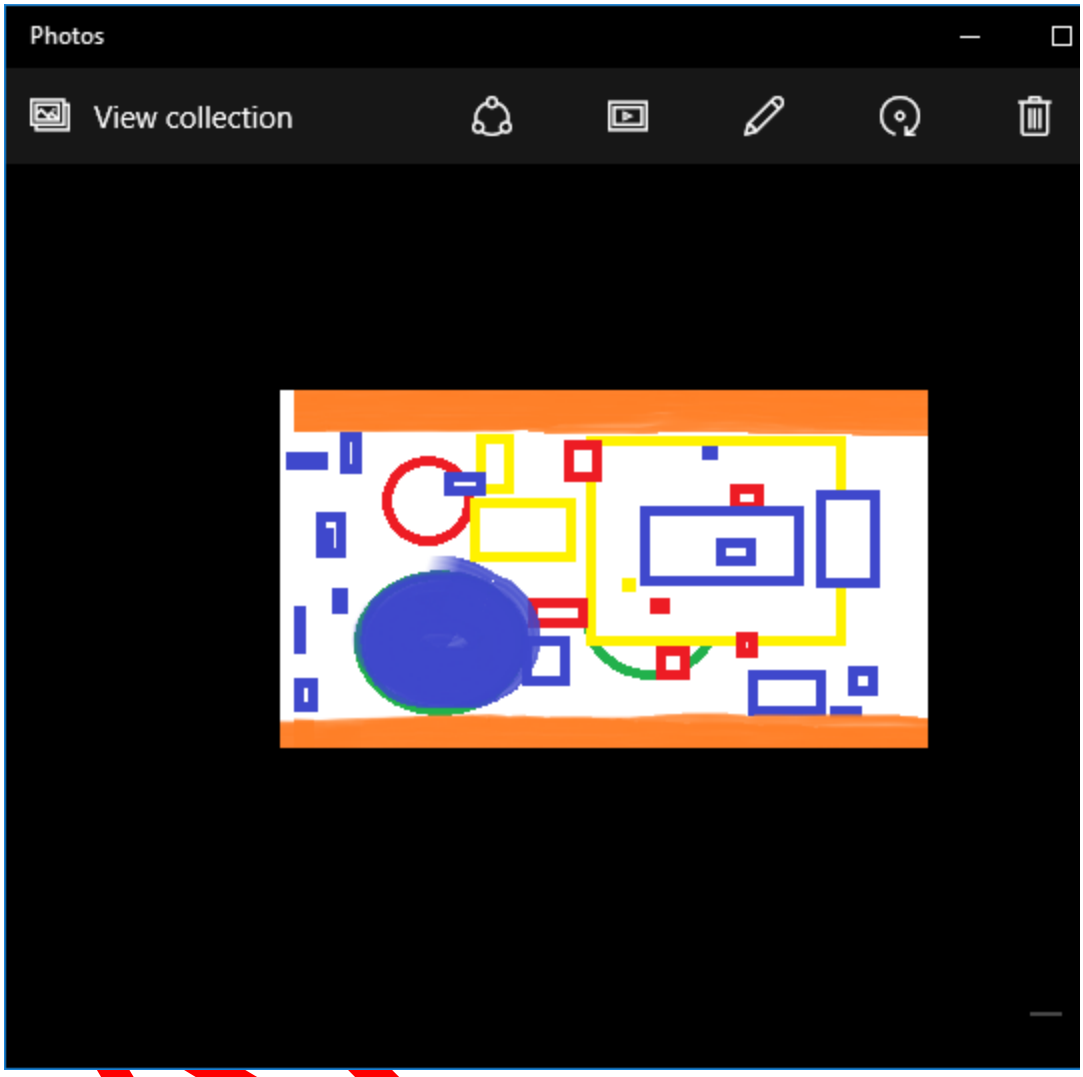

Figure 2.3: Content of SampleImage.png

Now let's store these two files in a SQL Server table.

```
USE master;
GO

--Create test database
CREATE DATABASE [BinaryFilesDB];
GO

--Use test database
USE [BinaryFilesDB];
GO

--Create table that will be hosting the files
CREATE TABLE [dbo].[tblFiles]
    (
      [fileID] [BIGINT] IDENTITY(1, 1) ,
      [fileName] [NVARCHAR](255) NULL ,
      [binFile] [IMAGE] NOT NULL
    );
GO

--
--Import SampleTextFile.txt
--
INSERT  INTO dbo.tblFiles
        ( [fileName] ,
          [binFile]
        )
VALUES  ( 'SampleTextFile.txt' ,
          ( SELECT  *
            FROM    OPENROWSET(BULK 'c:\testing\SampleTextFile.txt',
                               SINGLE_BLOB) AS x
          )
        );
GO
--
--Import SampleImage.png
--
INSERT  INTO dbo.tblFiles
        ( [fileName] ,
          [binFile]
        )
VALUES  ( 'SampleImage.png' ,
          ( SELECT  *
            FROM    OPENROWSET(BULK 'c:\testing\SampleImage.png', SINGLE_BLOB)
                    AS x
          )
        );
GO
```

Listing 2.1: Import Binary Files.

Let's check the contents of "**tblFiles**" table:

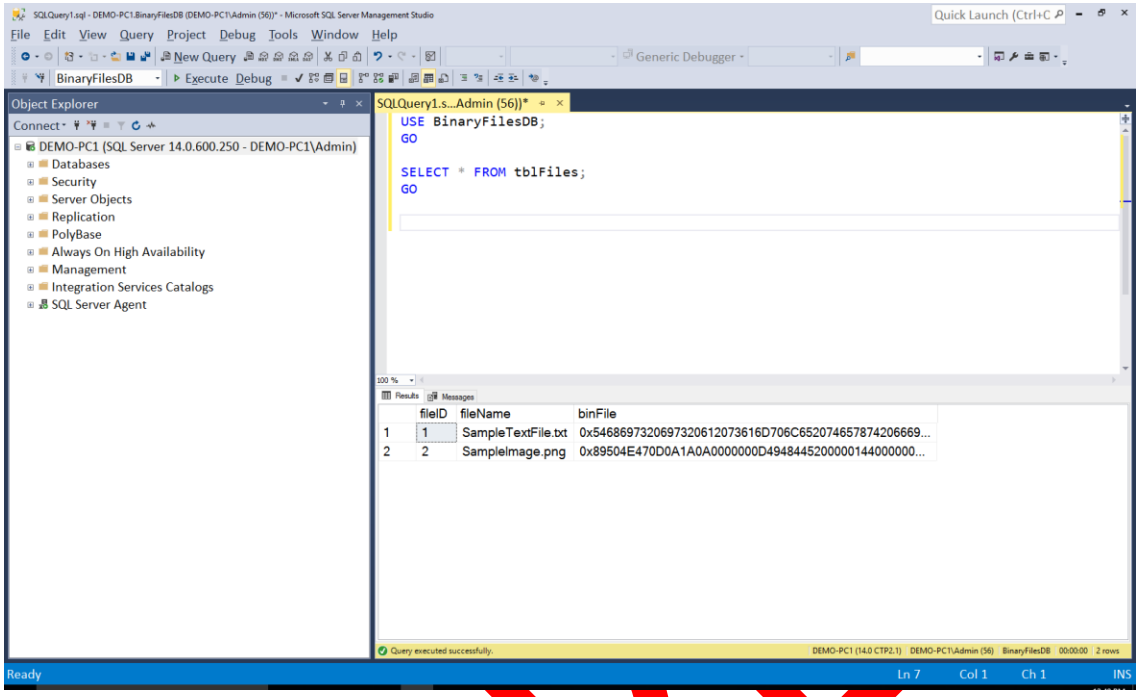

Figure 2.4: Contents of the table after importing the binary files.

Let's rename the files, just for performing a minor modification:

```
--Rename files
UPDATE   tblFiles
SET      fileName = 'SampleTextFileModified.txt'
WHERE    filename = 'SampleTextFile.txt';
GO

UPDATE   tblFiles
SET      fileName = 'SampleImageModified.png'
WHERE    filename = 'SampleImage.png';
GO
```

Listing 2.2: Rename Files Through Tables.

Let's check the table contents again:

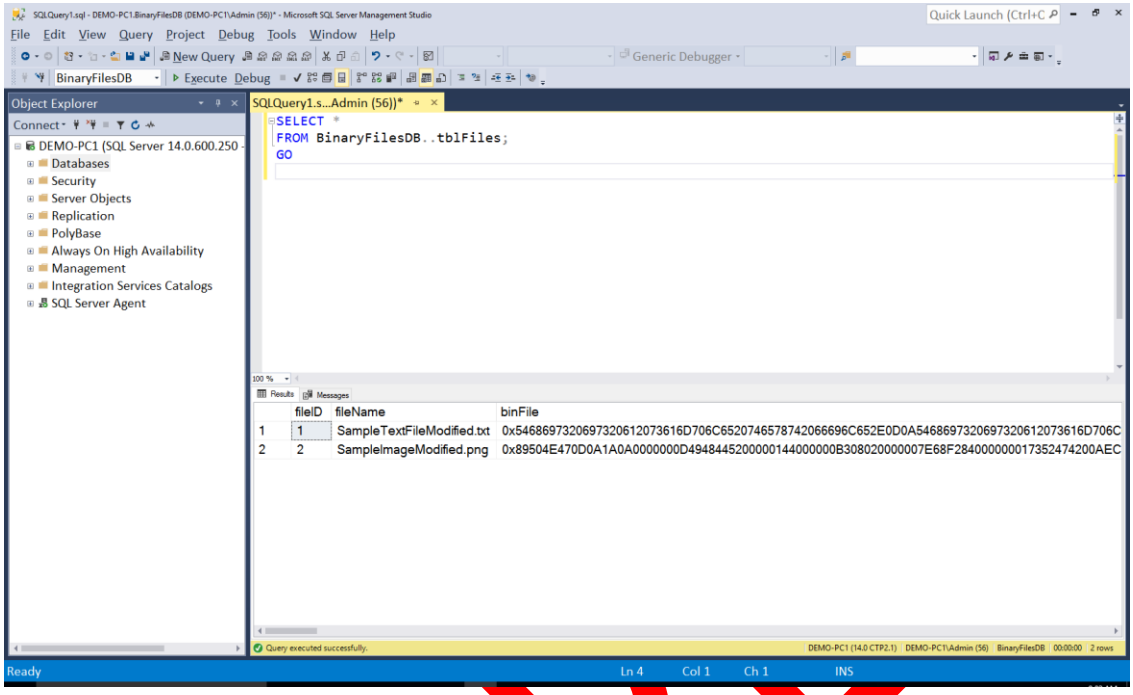

Figure 2.5: Renamed contents of table "tblFiles"

Now let's produce the export statements:

```
USE BinaryFilesDB;
GO

SELECT  'bcp "select binFile from BinaryFilesDB.dbo.tblFiles where fileid='
        + CAST (fileID AS VARCHAR(50)) + '" queryout "c:\testing\'
        + [filename] + '" -f "c:\testing\bcp.fmt" -S . -T' AS RunTheseOnCommandPrompt
FROM    BinaryFilesDB.dbo.tblFiles;
GO
```

Listing 2.3: Export Files from Tables.

Some notes on the above T-SQL statement:

- I'm using a default instance for SQL Server (.)
- Also note that I am using Trusted (-T) connection for my generated bcp commands
- If you want to use username/password instead of a Trusted connection replace "-T" with -U [username] -P [password]
- You can download the bcp.fmt format file from here.

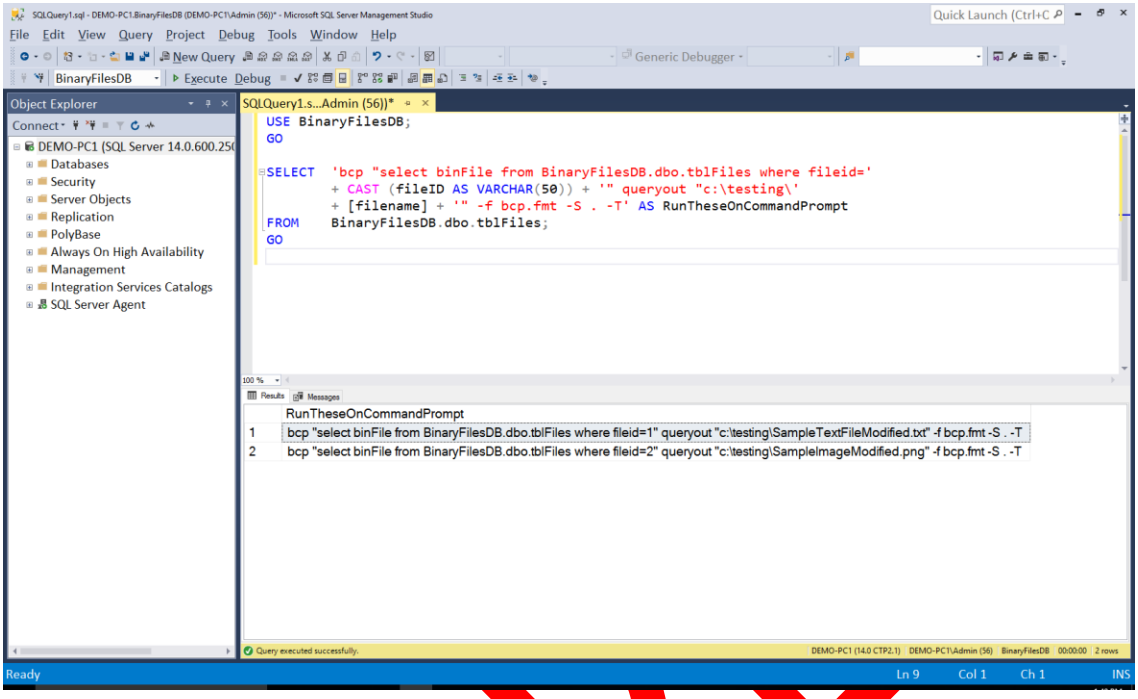

Figure 2.6: Generated statements for exporting the binary files.

Now let's run the two bcp commands on the command prompt:

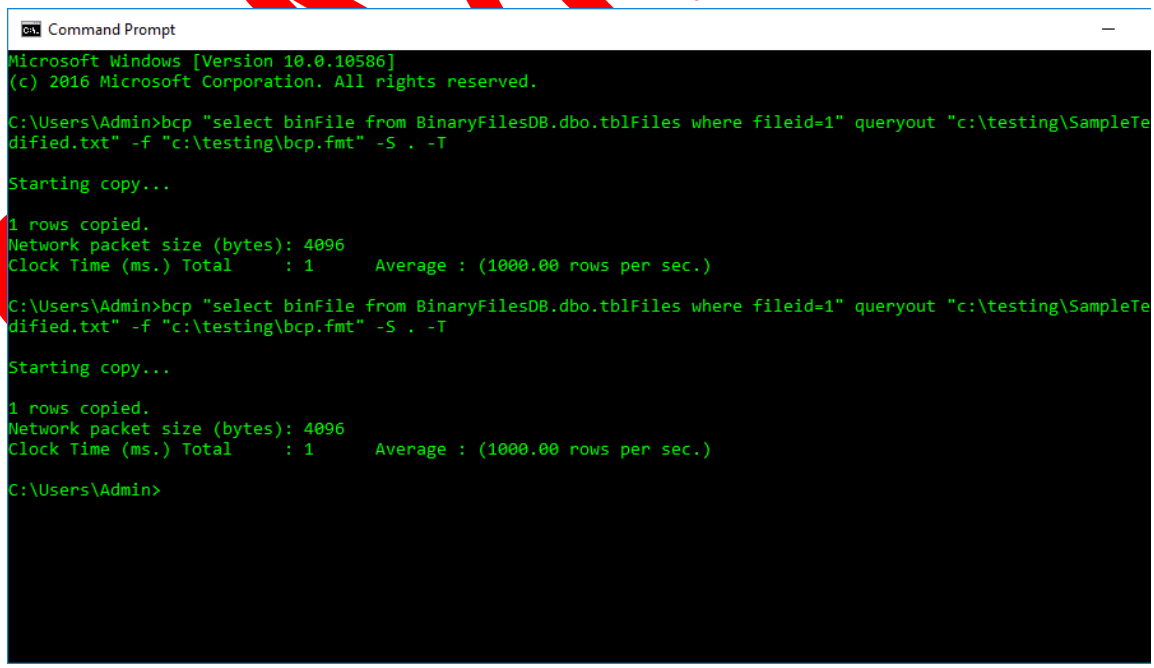

Figure 2.7: Executing the bcp commands for exporting the binary files.

Also, let's check the contents of the "c:\testing" directory:

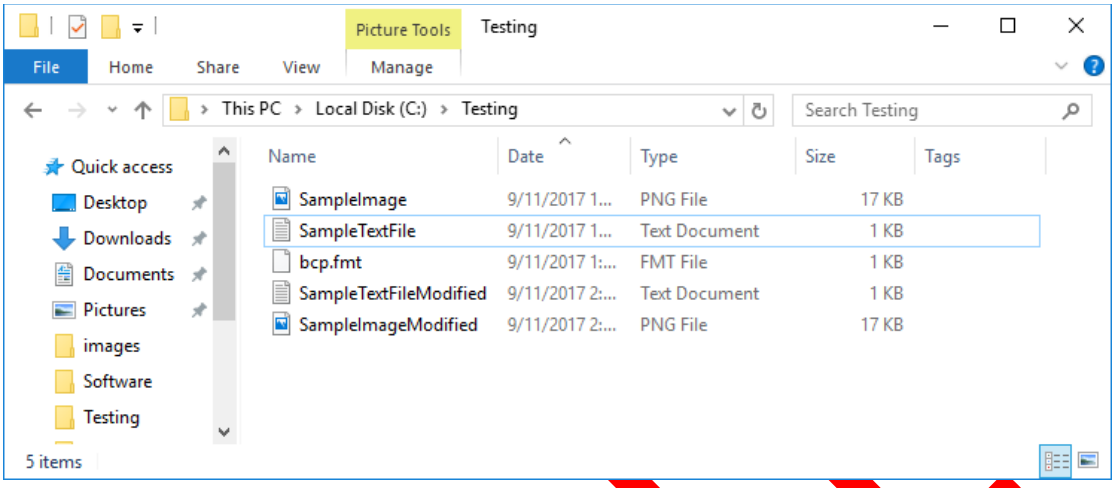

Figure 2.8: Binary files successfully exported from SQL Server.

As you can see, the two renamed files were successfully exported and have the exact same size as the original ones!

Moreover, their contents are exactly the same:

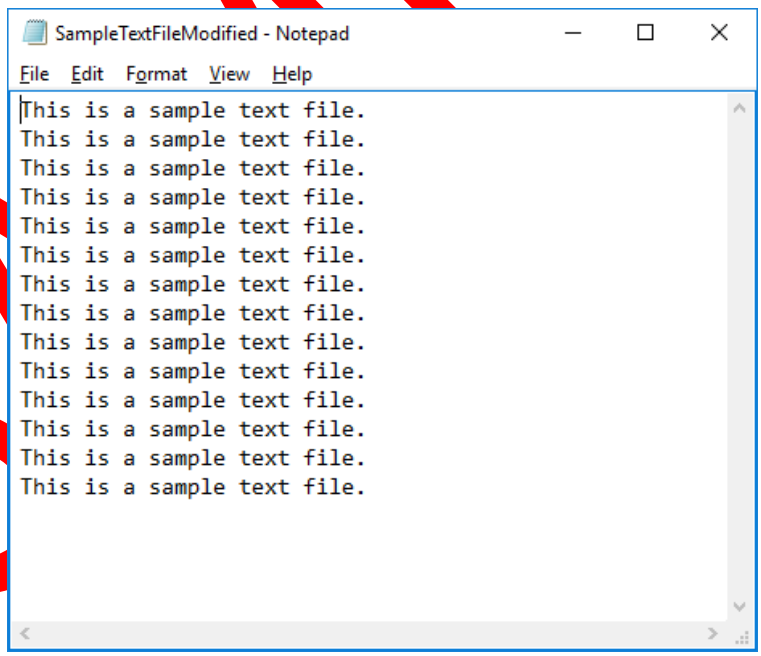

Figure 2.9: Contents of SampleTextFileModified.txt.

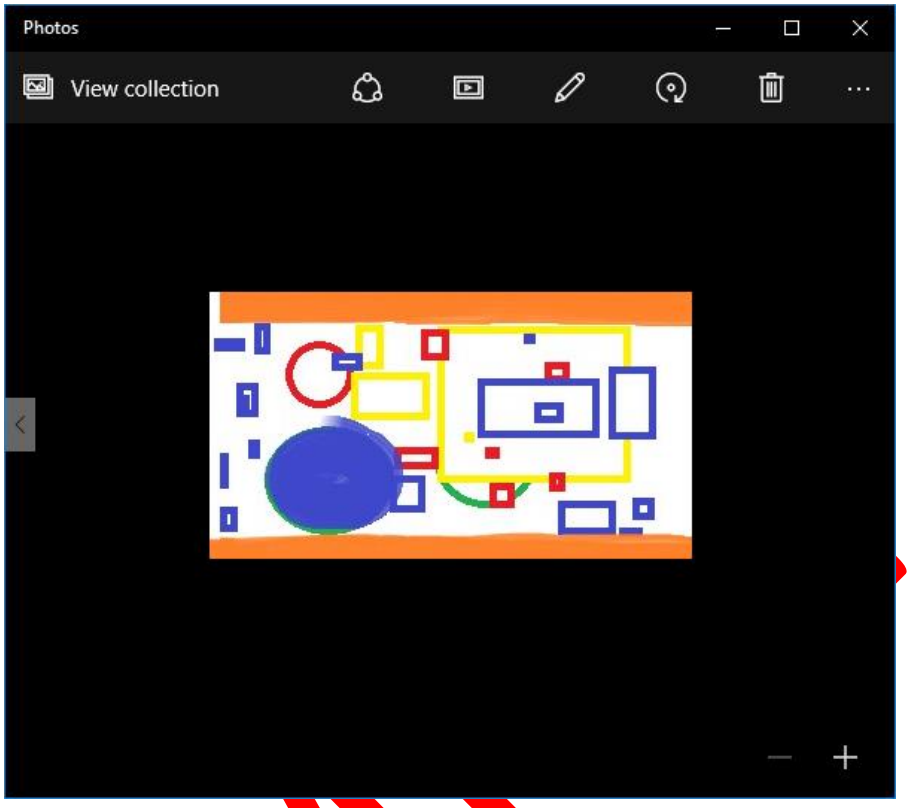

Figure 2.10: Contents of SampleImageFileModified.png.

➔ *Applies to: SQL Server 2005 or later.*

▪ How to Import and Export Unstructured Data in SQL Server - FILESTREAM

In the previous article, we discussed how we can import and export unstructured data in SQL Server by using the "image" data type.

In this article we will see how we can manipulate unstructured data in SQL Server 2008 or later by using the FILESTREAM feature. FILESTREAM allows storing unstructured data (i.e. music, video, documents, etc.) onto the NTFS file system and manipulating it via the Database Engine.

Step 1: Enabling FILESTREAM

Before using the FILESTREAM feature you have to enable it. To this end, you need to navigate to **SQL Server Configuration Manager** and under **SQL Server Services** right click on the SQL Server instance, select properties and in the "**FileStream**" tab check "**Enable FILESTREAM for Transact-SQL access**" and "**Enable FILESTREAM for file I/O access**":

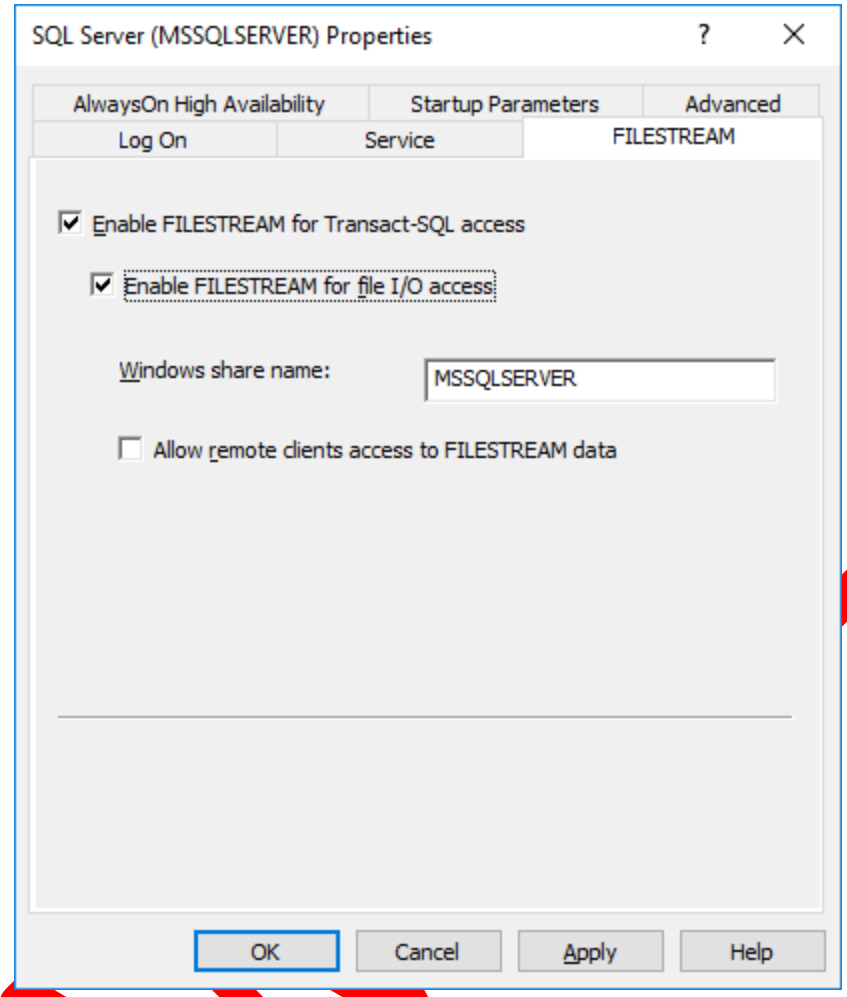

Figure 2.11: Enabling FILESTREAM.

The last step for enabling FILESTREAM is from within SSMS, to open a new query window and execute the following T-SQL statement and then restart the SQL Server instance:

```
EXEC sp_configure filestream_access_level, 2;
GO
RECONFIGURE;
GO
```

Listing 2.4: Enabling FILESTREAM.

Note: the available FILESTREAM access levels are:

- **0**: Disables FILESTREAM support for this instance.
- **1**: Enables FILESTREAM for Transact-SQL access.
- **2**: Enables FILESTREAM for Transact-SQL and Win32 streaming access.

Step 2: Creating a FILESTREAM-Enabled Database

For creating a FILESTREAM-enabled database you just need to include a FILESTREAM filegroup. For example:

```
CREATE DATABASE FileStreamDB ON PRIMARY ( NAME = FileStreamDBData,
    FILENAME = 'C:\Blog\SQLData\filestreamDB_data.mdf'), FILEGROUP
    FileStreamGroup_1 CONTAINS FILESTREAM( NAME = FileStreamDBFS,
    FILENAME = 'C:\Blog\SQLData\filestream1') LOG ON  ( NAME = FileStreamDBLogs,
    FILENAME = 'C:\Blog\SQLData\filestreamDB_log.ldf');
GO
```

Listing 2.5: Create a FILESTREAM-Enabled Database.

Step 3: Creating a Table for Storing FileStream Data

The only difference between a "normal" table and a table that can store filestream data is the use of the "FILESTREAM" data type for a specific column in the table's definition script:

```
USE [FileStreamDB];
GO

CREATE TABLE dbo.Files
    (
      [Id] [UNIQUEIDENTIFIER] ROWGUIDCOL
                              NOT NULL
                              UNIQUE ,
      [FileName] VARCHAR(100) ,
      [ActualFile] VARBINARY(MAX) FILESTREAM
                                  NULL
    );
GO
```

Listing 2.6: Create Table for Storing FILESTREAM Data.

Step 4: Storing FileStream Data

For this example, consider the following unstructured data file (image):

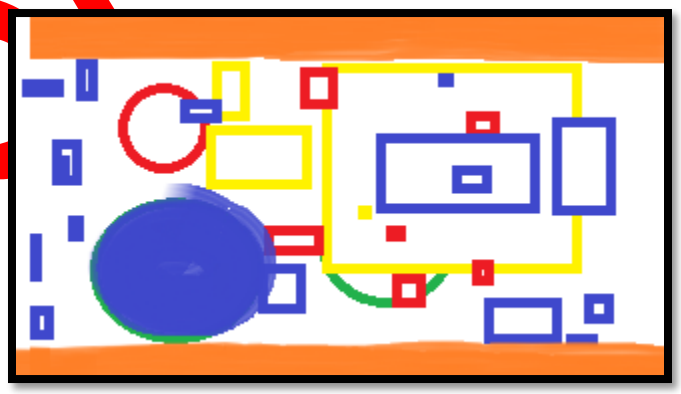

Figure 2.12: Image file to be stored in FILESTREAM-enabled database.

Now, let's store the file in our FILESTREAM-enabled database and table that was created earlier:

```
USE [FileStreamDB];
GO

INSERT  INTO dbo.Files
VALUES  ( NEWID(), 'SampleImage.png',
          ( SELECT  *
            FROM    OPENROWSET(BULK 'C:\Testing\2\SampleImage.png',
                               SINGLE_BLOB) AS x
          ) );
GO
```

Listing 2.7: Store File into the FILESTEAM-Database.

Here are the contents of the table:

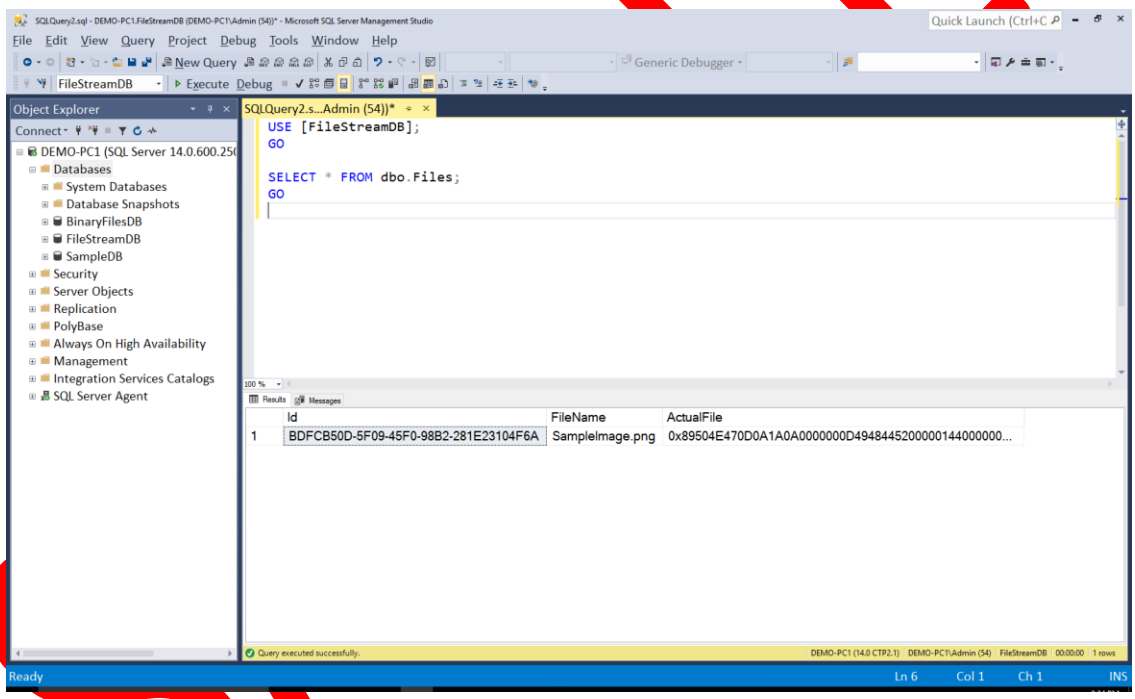

Figure 2.13: The contents of the FILESTREAM-enabled table after inserting unstructured data (image file).

As you can see, the file is visible on the file system level too:

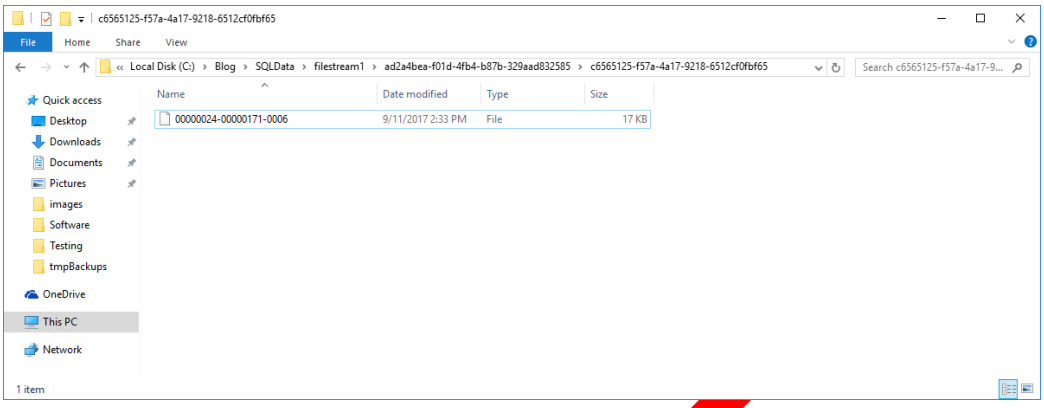

Figure 2.14: Binary file stored using FILESTREAM - Accessible on the file system level.

Now let's try to open the file using MS Paint:

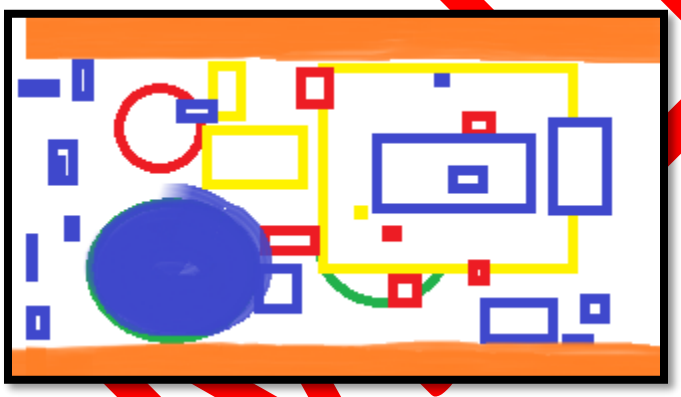

Figure 2.15: Accessing the data stored in the FILESTREAM-enabled database from the file system level (Windows).

As you can see, the image file is stored in the SQL Server database table but besides T-SQL access, you can also access it from Windows!

What we just did with the above example shows a small glimpse of the real power of FILESTREAM that is leveraging the performance and rich APIs of the Windows file system and at the same time maintaining consistency between structured and unstructured data in SQL Server.

FILESTREAM actually works like a bridge between structured and unstructured data via a combination of transactional data and file system access and can be extremely useful in cases where you have many binary objects like images and videos and you want to store it in SQL Server and being able to access it with the speed of the file system.

➔ ***Applies to: SQL Server 2008 or later.***

How to Import and Export Unstructured Data in SQL Server - FileTables

The first article in this chapter, explained how you can store and export binary files in earlier versions of SQL Server such as SQL Server 2005 with the use of the image datatype.

The second article presented the FILESTREAM technology which was first introduced in SQL Server 2008.

This article discusses the FileTables feature which builds on top of SQL Server's FILESTREAM technology. FileTables was first introduced in SQL Server 2012.

The FileTables feature allows the user to store unstructured data (i.e. files, documents, images, etc.) in special tables in SQL Server called FileTables, but being able to access them from the file system. To this end, if you have an application that needs to access unstructured data, you can directly access them from the File System even though the data is stored into FileTables in SQL Server.

Enough with the talking, let's see an example of using this great feature.

First, let's enable FILESTREAM (this is a prerequisite - for more info, see the previous article):

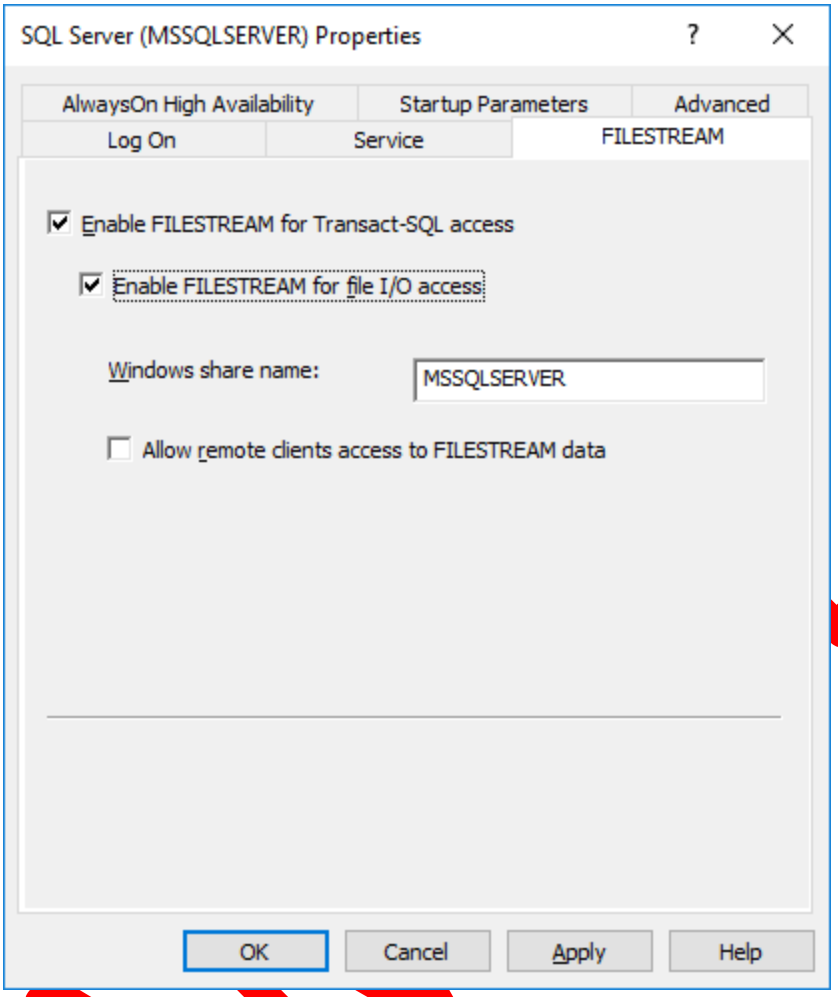

Figure 2.16: Enable FILESTREAM through SQL Instance Properties.

Then, let's create a FILESTREAM-enabled database (make sure to update the path "**C:\Blog\SQLData**\" if you are using a different one):

```
CREATE DATABASE FileStreamDB ON PRIMARY ( NAME = FileStreamDBData,
              FILENAME = 'C:\Blog\SQLData\filestreamDB_data.mdf'),
    FILEGROUP FileStreamGroup_1 CONTAINS FILESTREAM( NAME = FileStreamDBFS,
    FILENAME = 'C:\Blog\SQLData\filestream1') LOG ON  ( NAME = FileStreamDBLogs,

                    FILENAME = 'C:\Blog\SQLData\filestreamDB_log.ldf');
GO
```

Listing 2.8: DDL for FILESTREAM-Enabled Database.

Then we need to enable non-transactional access at the database level as well as specify the directory for FileTables:

```
ALTER DATABASE FileStreamDB
    SET FILESTREAM ( NON_TRANSACTED_ACCESS = FULL, DIRECTORY_NAME =
N'FileTablesDir')
GO
```

Listing 2.9: Enable Non-Transactional Access at Database-Level.

Now it's time to create the FileTable:

```
USE FileStreamDB;
GO

CREATE TABLE DocumentStore AS FILETABLE
    WITH (
         FILETABLE_DIRECTORY = 'FileTablesDir',
         FILETABLE_COLLATE_FILENAME = database_default
         );
GO
```

Listing 2.10: Create FileTable.

OK, it's time for the magic to take place. Let's check the contents of the FileTable:

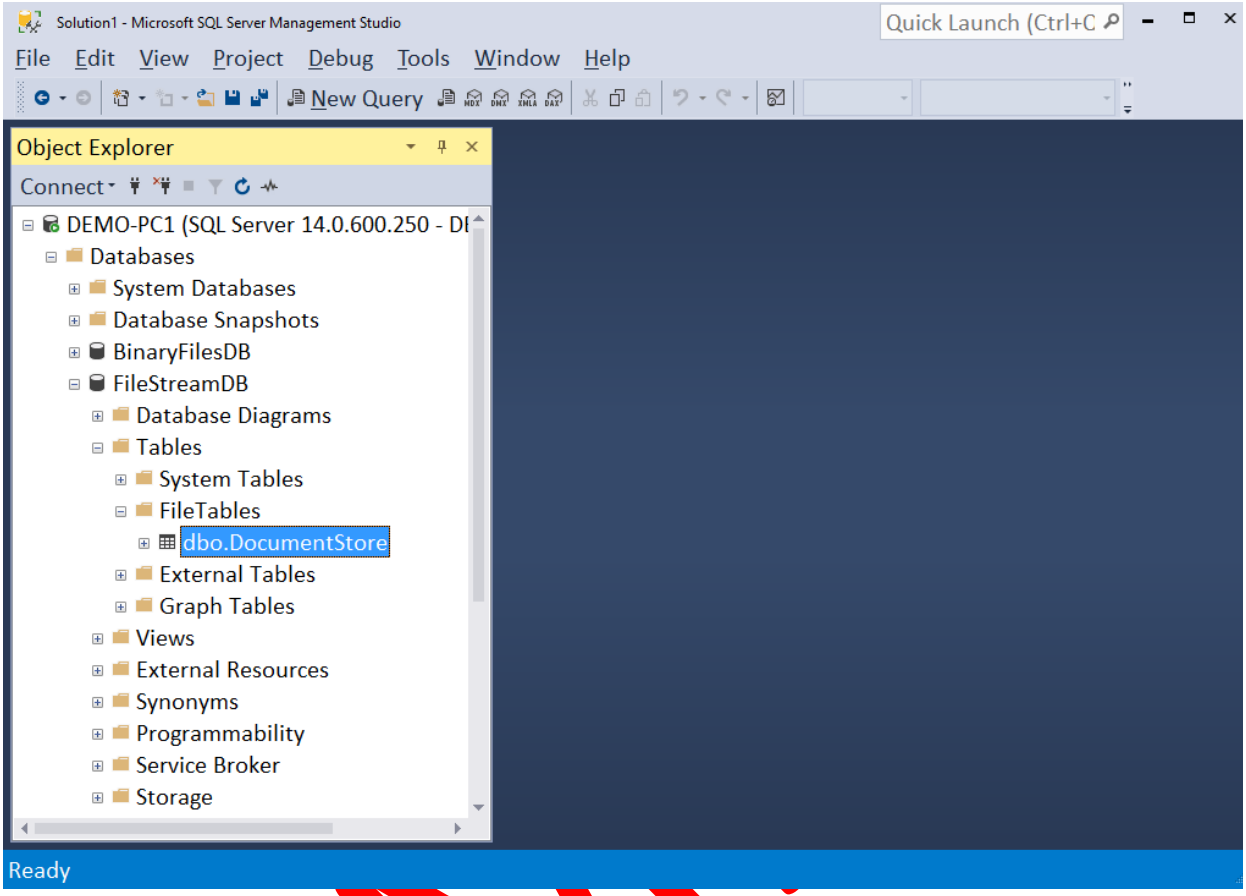

Figure 2.17: The FileTable in SSMS Object Explorer.

As you can see, the FileTable "DocumentStore" is currently empty:

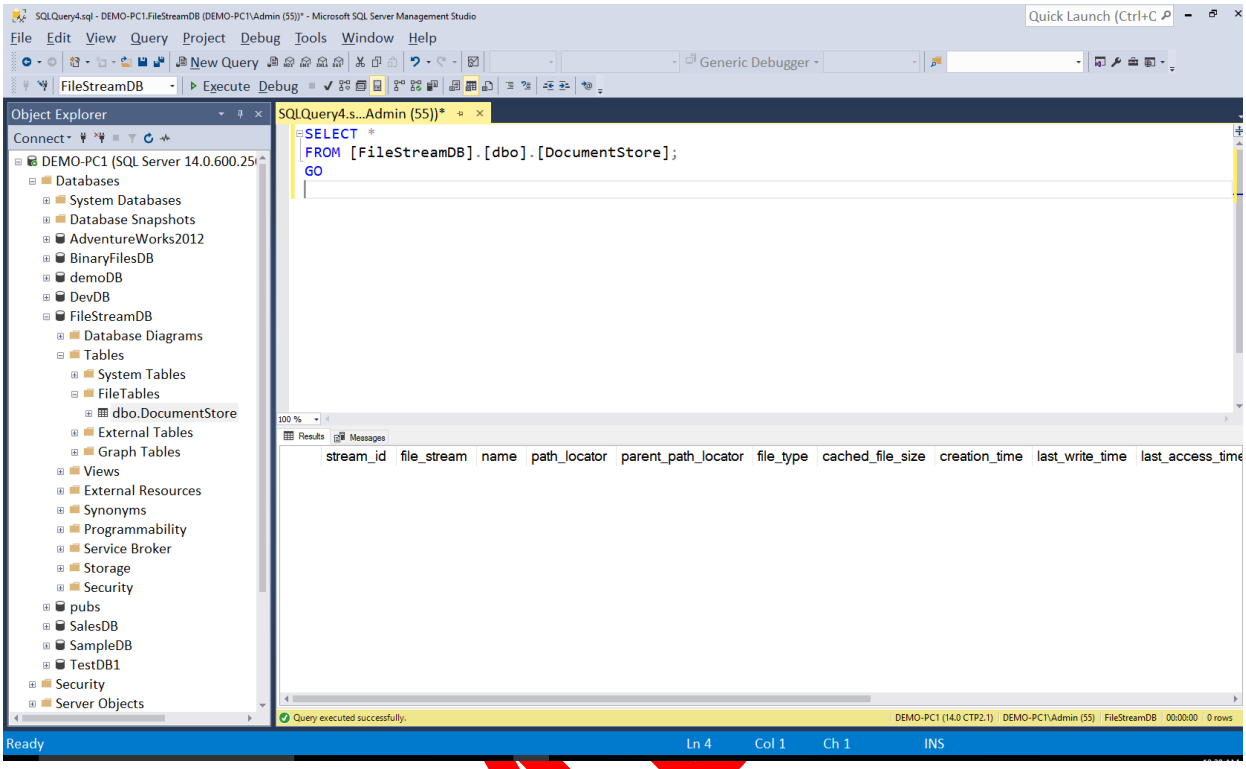

Figure 2.18: Checking the contents of the FileTable.

Now, let's open the FileTable directory in Explorer, and by accessing it directly from the Windows File System, copy some files:

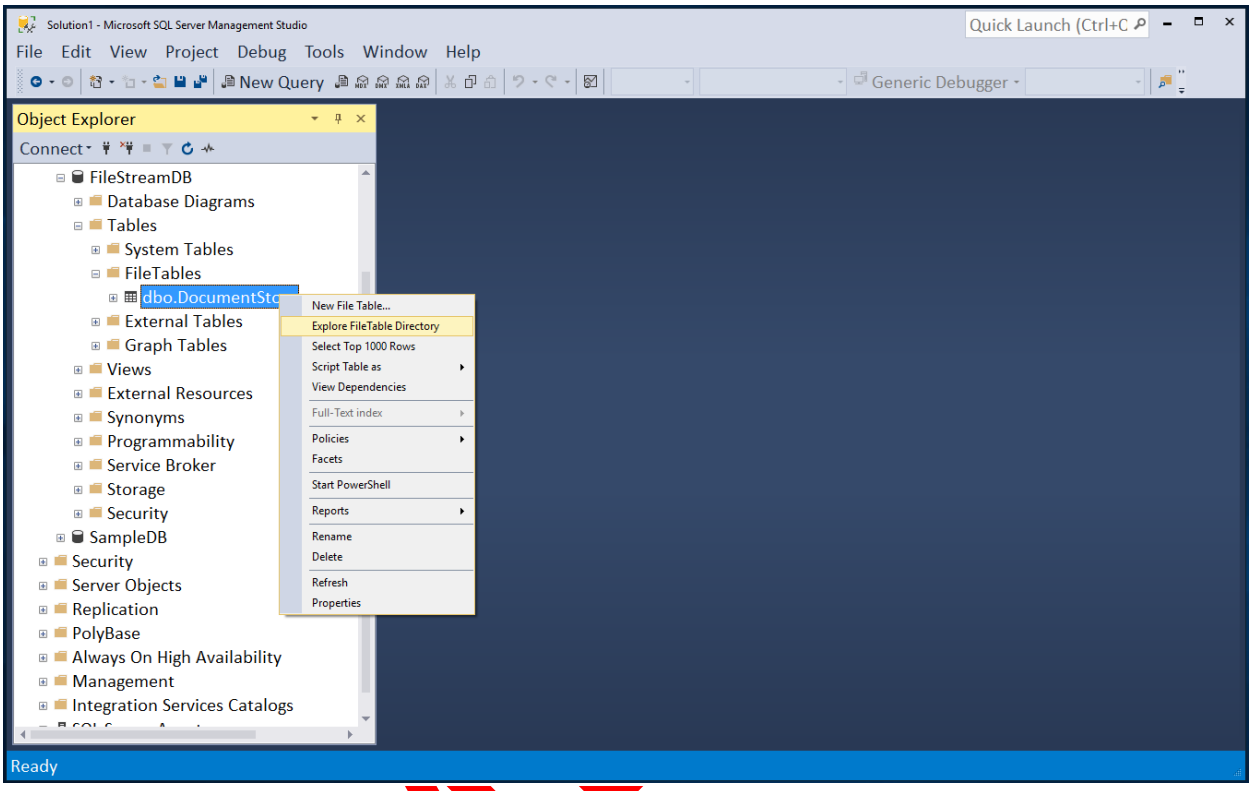

Figure 2.19: Access the FileTable directory in Explorer.

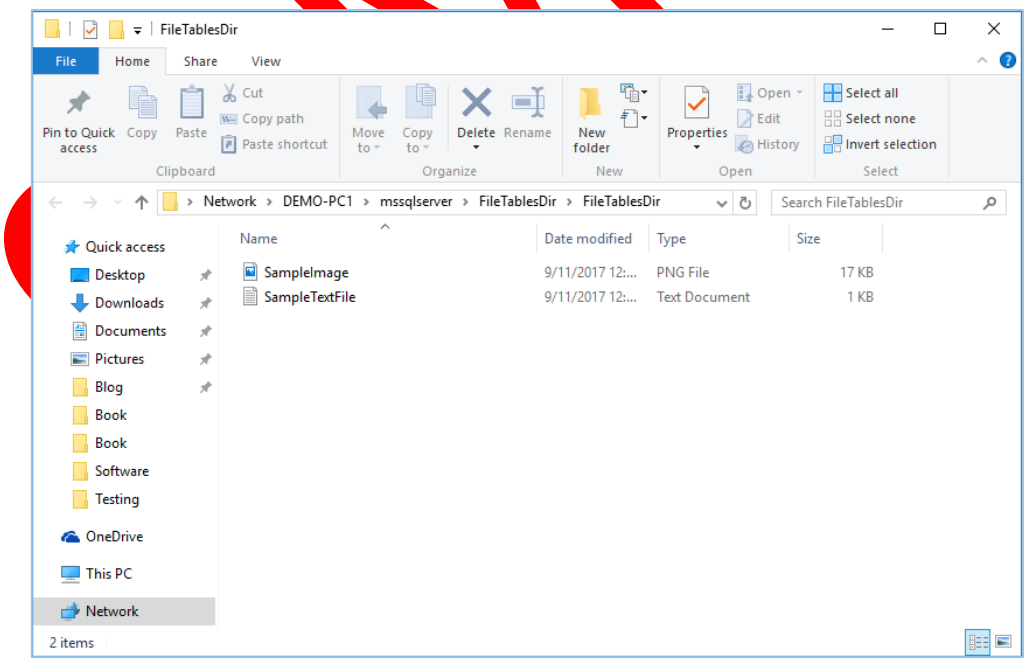

Figure 2.20: Copy files in the FileTable directory.

Now, let's check again the contents of the FileTable from SSMS:

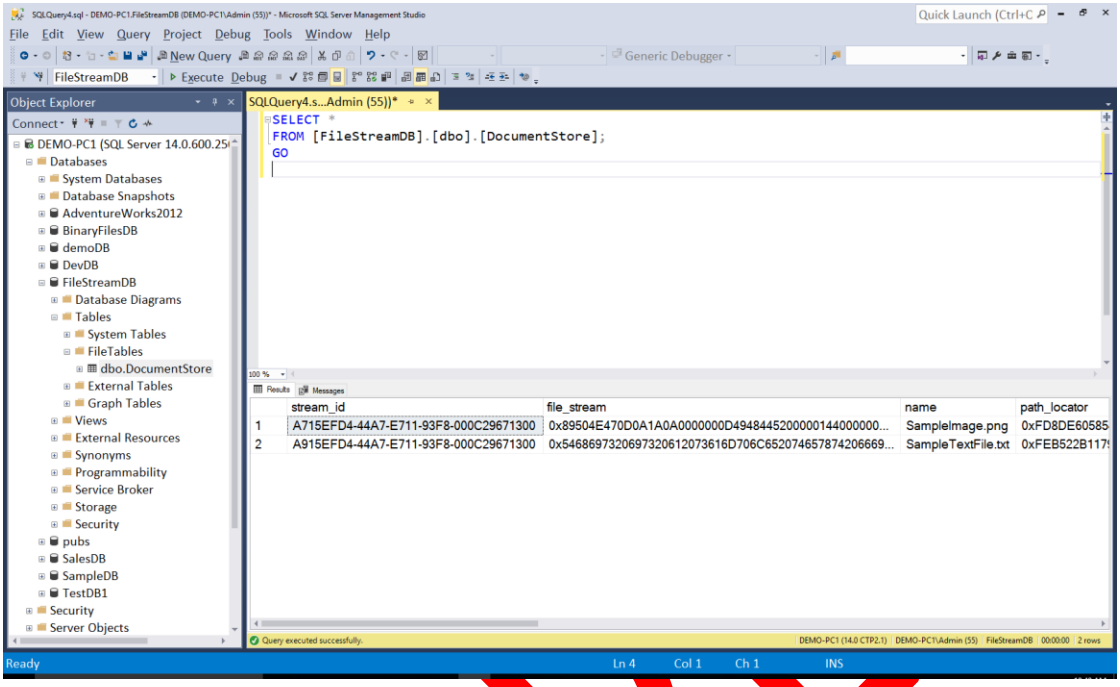

Figure 2.21: Checking the FileTable contents after copying the files from Windows Explorer.

And that's it! As you can see in the above screenshot, by copying the two files from Windows Explorer directly into the FileTable directory, the corresponding two records have been created in the FileTable in the SQL Server instance! Now you can access the files either from the SQL Server Database Engine via T-SQL or directly from the Windows File System!

FileTables along with the FILESTREAM technology enables the user with more options when it comes to storing binary objects in SQL Server. The seamless integration of FileTables with the Windows File System (NTFS) allows the user to store large binary objects in fast speeds and at the same time to be able to use the powerful features of SQL Server's Database Engine to traverse this data (i.e. full-text search, semantic search, etc.).

➔ *Applies to: SQL Server 2012 or later.*

▪ Summary

In this chapter, we learned how to work with unstructured data in different versions of SQL Server. We discussed how we could achieve that by utilizing the available features in each one of the SQL Server versions.

In the first article we discussed how we can import and export unstructured data in older versions of SQL Server (i.e. SQL Server 2005) using the IMAGE data type. In the second article,

we saw how we can do the same in SQL Server 2008 or later by using the FILESTREAM feature. Last, in the third article, we examined how we can import and export unstructured data using the FileTables feature available in SQL Server 2012 or later.

In the next chapter, we are going to talk about Data Access focusing on the ADO.NET Entity Framework and WCF Data Services.