# Binary Large Object (Blob) Data

SQL Server provides solutions for storing files and documents in the database or on remote storage devices.

## Options for Storing Blobs

### [FILESTREAM (SQL Server)](https://docs.microsoft.com/en-us/sql/relational-databases/blob/filestream-sql-server)

FILESTREAM enables SQL Server-based applications to store unstructured data, such as documents and images, on the file system. Applications can leverage the rich streaming APIs and performance of the file system and at the same time maintain transactional consistency between the unstructured data and corresponding structured data.

### [FileTables (SQL Server)](https://docs.microsoft.com/en-us/sql/relational-databases/blob/filetables-sql-server)

The FileTable feature brings support for the Windows file namespace and compatibility with Windows applications to the file data stored in SQL Server. FileTable lets an application integrate its storage and data management components, and provides integrated SQL Server services - including full-text search and semantic search - over unstructured data and metadata.

In other words, you can store files and documents in special tables in SQL Server called FileTables, but access them from Windows applications as if they were stored in the file system, without making any changes to your client applications.

### [Remote Blob Store (RBS)](https://docs.microsoft.com/en-us/sql/relational-databases/blob/remote-blob-store-rbs-sql-server)

Remote BLOB store (RBS) for SQL Server lets database administrators store binary large objects (BLOBs) in commodity storage solutions instead of directly on the server. This saves a significant amount of space and avoids wasting expensive server hardware resources. RBS provides a set of API libraries that define a standardized model for applications to access BLOB data. RBS also includes maintenance tools, such as garbage collection, to help manage remote BLOB data.

RBS is included on the SQL Server installation media, but is not installed by the SQL Server Setup program.

## Why RBS?

### Optimized database storage and performance

Storing BLOBs in the database can consume large amounts of file space and expensive server resources. RBS transfers the BLOBs to a dedicated storage solution you choose and stores references to thr BLOBs in the database. This frees server storage for structured data, and frees server resources for database operations.

### Efficient BLOB management

Several RBS features support stored BLOBs management:

* BLOBS are managed with ACID (atomic consistency isolation durable) transactions.
* BLOBs are organized into collections.
* Garbage collection, consistency checking, and other maintenance functions are included.

### Comparing FILESTREAM and FileTable

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **File Server and Database Solution** | **FILESTREAM Solution** | **FileTable Solution** |
| **Single story for management tasks** | No | Yes | Yes |
| **Single set of services: search, reporting, querying, and so forth** | No | Yes | Yes |
| **Integrated security model** | No | Yes | Yes |
| **In-place updates of FILESTREAM data** | Yes | No | Yes |
| **File and directory hierarchy maintained in the database** | No | No | Yes |
| **Windows application compatibility** | Yes | No | Yes |
| **Relational access to file attributes** | No | No | Yes |

FILESTREAM enables SQL Server-based applications to store unstructured data, such as documents and images, on the file system. Applications can leverage the rich streaming APIs and performance of the file system and at the same time maintain transactional consistency between the unstructured data and corresponding structured data.

FILESTREAM integrates the SQL Server Database Engine with an NTFS or ReFS file systems by storing **varbinary(max)** binary large object (BLOB) data as files on the file system. Transact-SQL statements can insert, update, query, search, and back up FILESTREAM data. Win32 file system interfaces provide streaming access to the data.

FILESTREAM uses the NT system cache for caching file data. This helps reduce any effect that FILESTREAM data might have on Database Engine performance. The SQL Server buffer pool is not used; therefore, this memory is available for query processing.

FILESTREAM is not automatically enabled when you install or upgrade SQL Server. You must enable FILESTREAM by using SQL Server Configuration Manager and SQL Server Management Studio. To use FILESTREAM, you must create or modify a database to contain a special type of filegroup. Then, create or modify a table so that it contains a **varbinary(max)** column with the FILESTREAM attribute. After you complete these tasks, you can use Transact-SQL and Win32 to manage the FILESTREAM data.

## When to Use FILESTREAM

In SQL Server, BLOBs can be standard **varbinary(max)** data that stores the data in tables, or FILESTREAM **varbinary(max)** objects that store the data in the file system. The size and use of the data determines whether you should use database storage or file system storage. If the following conditions are true, you should consider using FILESTREAM:

* Objects that are being stored are, on average, larger than 1 MB.
* Fast read access is important.
* You are developing applications that use a middle tier for application logic.

For smaller objects, storing **varbinary(max)** BLOBs in the database often provides better streaming performance.

## FILESTREAM Storage

FILESTREAM storage is implemented as a **varbinary(max)** column in which the data is stored as BLOBs in the file system. The sizes of the BLOBs are limited only by the volume size of the file system. The standard **varbinary(max)** limitation of 2-GB file sizes does not apply to BLOBs that are stored in the file system.

To specify that a column should store data on the file system, specify the FILESTREAM attribute on a **varbinary(max)** column. This causes the Database Engine to store all data for that column on the file system, but not in the database file.

FILESTREAM data must be stored in FILESTREAM filegroups. A FILESTREAM filegroup is a special filegroup that contains file system directories instead of the files themselves. These file system directories are called data containers. Data containers are the interface between Database Engine storage and file system storage.

When you use FILESTREAM storage, consider the following:

* When a table contains a FILESTREAM column, each row must have a nonnull unique row ID.
* Multiple data containers can be added to a FILESTREAM filegroup.
* FILESTREAM data containers cannot be nested.
* When you are using failover clustering, the FILESTREAM filegroups must be on shared disk resources.
* FILESTREAM filegroups can be on compressed volumes.

### Accessing BLOB Data with Transact-SQL and File System Streaming Access

After you store data in a FILESTREAM column, you can access the files by using Transact-SQL transactions or by using Win32 APIs.

### Transact-SQL Access

By using Transact-SQL, you can insert, update, and delete FILESTREAM data:

* You can use an insert operation to prepopulate a FILESTREAM field with a null value, empty value, or relatively short inline data. However, a large amount of data is more efficiently streamed into a file that uses Win32 interfaces.
* When you update a FILESTREAM field, you modify the underlying BLOB data in the file system. When a FILESTREAM field is set to NULL, the BLOB data associated with the field is deleted. You cannot use a Transact-SQL chunked update, implemented as UPDATE**.**Write(), to perform partial updates to the data.
* When you delete a row or delete or truncate a table that contains FILESTREAM data, you delete the underlying BLOB data in the file system.

#### Enable and configure FILESTREAM

EXEC sp\_configure filestream\_access\_level, 2

RECONFIGURE

#### To create a FILESTREAM-enabled database

How to create a database that supports FILESTREAM. Because FILESTREAM uses a special type of filegroup, when you create the database, you must specify the CONTAINS FILESTREAM clause for at least one filegroup. A FILESTREAM filegroup can contain more than one file.

CREATE DATABASE Archive

ON

PRIMARY ( NAME = Arch1,

FILENAME = 'c:\data\archdat1.mdf'),

FILEGROUP FileStreamGroup1 CONTAINS FILESTREAM( NAME = Arch3,

FILENAME = 'c:\data\filestream1')

LOG ON ( NAME = Archlog1,

#### Create a Table for Storing FILESTREAM Data

When the database has a FILESTREAM filegroup, you can create or modify tables to store FILESTREAM data. To specify that a column contains FILESTREAM data, you create a **varbinary(max)** column and add the FILESTREAM attribute.

CREATE TABLE Archive.dbo.Records

( [Id] [uniqueidentifier] ROWGUIDCOL NOT NULL UNIQUE,

[SerialNumber] INTEGER UNIQUE,

[Chart] VARBINARY(MAX) FILESTREAM NULL

)

GO

#### Access FILESTREAM Data with Transact-SQL

## Inserting a Row That Contains FILESTREAM Data

**Inserting NULL**

INSERT INTO Archive.dbo.Records VALUES (newid (), 1, NULL);

**Inserting a Zero-Length Record**

INSERT INTO Archive.dbo.Records VALUES (newid (), 2,CAST ('' as varbinary(max)));

**Creating a Data File**

The following example shows how to use INSERT to create a file that contains data. The Database Engine converts the string Seismic Data to a varbinary(max) value. FILESTREAM creates the Windows file if it does not already exist.The data is then added to the data file.

SQLCopy

INSERT INTO Archive.dbo.Records

VALUES (newid (), 3,

CAST ('Seismic Data' as varbinary(max)));

GO

When you select all data from the Archive.dbo.Records table, the results are similar to the results that are shown in the following table. However, the Id column will contain different GUIDs.

| Id | SerialNumber | Chart |
| --- | --- | --- |
| C871B90F-D25E-47B3-A560-7CC0CA405DAC | 1 | NULL |
| F8F5C314-0559-4927-8FA9-1535EE0BDF50 | 2 | 0x |
| 7F680840-B7A4-45D4-8CD5-527C44D35B3F | 3 | 0x536569736D69632044617461 |

**Updating FILESTREAM Data**

UPDATE Archive.dbo.Records SET [Chart] = CAST('Xray 1' as varbinary(max))

WHERE [SerialNumber] = 2;

**Deleting FILESTREAM Data**

DELETE Archive.dbo.Records WHERE SerialNumber = 1;

#### Create Client Applications for FILESTREAM Data

#### Access FILESTREAM Data with OpenSqlFilestream

#### Make Partial Updates to FILESTREAM Data

#### FILESTREAM DDL, Functions, Stored Procedures, and Views

#### System Functions

* [GET\_FILESTREAM\_TRANSACTION\_CONTEXT (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/functions/get-filestream-transaction-context-transact-sql)
* [PathName (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-functions/pathname-transact-sql)

#### System Stored Procedures

* [sp\_configure (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-configure-transact-sql)
* [sp\_filestream\_force\_garbage\_collection (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/filestream-and-filetable-sp-filestream-force-garbage-collection)

#### System Views – Catalog Views

* [sys.database\_filestream\_options (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-database-filestream-options-transact-sql)

#### System Views – Dynamic Management Views

* [sys.dm\_filestream\_file\_io\_handles (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-filestream-file-io-handles-transact-sql)
* [sys.dm\_filestream\_file\_io\_requests (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-filestream-file-io-requests-transact-sql)

**FileTables**

The FileTable feature brings support for the Windows file namespace and compatibility with Windows applications to the file data stored in SQL Server. FileTable lets an application integrate its storage and data management components, and provides integrated SQL Server services - including full-text search and semantic search - over unstructured data and metadata.+

In other words, you can store files and documents in special tables in SQL Server called FileTables, but access them from Windows applications as if they were stored in the file system, without making any changes to your client applications.

The FileTable feature builds on top of SQL Server FILESTREAM technology.

## Benefits of the FileTable Feature

The goals of the FileTable feature include the following:

* Windows API compatibility for file data stored within a SQL Server database. Windows API compatibility includes the following:
  + Non-transactional streaming access and in-place updates to FILESTREAM data.
  + A hierarchical namespace of directories and files.
  + Storage of file attributes, such as created date and modified date.
  + Support for Windows file and directory management APIs.
* Compatibility with other SQL Server features including management tools, services, and relational query capabilities over FILESTREAM and file attribute data.

Thus FileTables remove a significant barrier to the use of SQL Server for the storage and management of unstructured data that is currently residing as files on file servers. Enterprises can move this data from file servers into FileTables to take advantage of integrated administration and services provided by SQL Server. At the same time, they can maintain Windows application compatibility for their existing Windows applications that see this data as files in the file system.

## What Is a FileTable?

SQL Server provides a special **table of files**, also referred to as a **FileTable**, for applications that require file and directory storage in the database, with Windows API compatibility and non-transactional access. A FileTable is a specialized user table with a pre-defined schema that stores FILESTREAM data, as well as file and directory hierarchy information and file attributes.

A FileTable provides the following functionality:

* A FileTable represents a hierarchy of directories and files. It stores data related to all the nodes in that hierarchy, for both directories and the files they contain. This hierarchy starts from a root directory that you specify when you create the FileTable.
* Every row in a FileTable represents a file or a directory.
* Every row contains the following items. For more information about the schema of a FileTable, see [FileTable Schema](https://docs.microsoft.com/en-us/sql/relational-databases/blob/filetable-schema).
  + A **file\_stream** column for stream data and a **stream\_id** (GUID) identifier. (The **file\_stream** column is NULL for a directory.)
  + Both **path\_locator** and **parent\_path\_locator** columns for representing and maintaining the current item (file or directory) and directory hierarchy.
  + 10 file attributes such as created date and modified date that are useful with file I/O APIs.
  + A type column that supports full-text search and semantic search over files and documents.
* A FileTable enforces certain system-defined constraints and triggers to maintain file namespace semantics.
* When the database is configured for non-transactional access, the file and directory hierarchy represented in the FileTable is exposed under the FILESTREAM share configured for the SQL Server instance. This provides file system access for Windows applications.

**Some additional characteristics of FileTables include the following:**

* The file and directory data stored in a FileTable is exposed through a Windows share for non-transactional file access for Windows API based applications. For a Windows application, this looks like a normal share with its files and directories. Applications can use a rich set of Windows APIs to manage the files and directories under this share.
* The directory hierarchy surfaced through the share is a purely logical directory structure that is maintained within the FileTable.
* Calls to create or change a file or directory through the Windows share are intercepted by a SQL Server component and reflected in the corresponding relational data in the FileTable.
* Windows API operations are non-transactional in nature, and are not associated with user transactions. However, transactional access to FILESTREAM data stored in a FileTable is fully supported, as is the case for any FILESTREAM column in a regular table.
* FileTables can also be queried and updated through normal Transact-SQL access. They are also integrated with SQL Server management tools, and features such as backup.

**Create, Alter, and Drop FileTables**

CREATE TABLE DocumentStore AS FileTable

WITH ( FileTable\_Directory = 'DocumentTable',

FileTable\_Collate\_Filename = database\_default );

**Altering a FileTable**

ALTER TABLE filetable\_name SET ( FILETABLE\_DIRECTORY = N'directory\_name' );

**Dropping a FileTable**

You can drop a FileTable by using the ordinary syntax for the [DROP TABLE](https://docs.microsoft.com/en-us/sql/t-sql/statements/drop-table-transact-sql)

**Load Files into FileTables**

-- Add a path locator column to the PhotoMetadata table.

ALTER TABLE PhotoMetadata ADD pathlocator hierarchyid;

-- Get the root path of the Photo directory on the File Server.

DECLARE @UNCPathRoot varchar(100) = '\\RemoteShare\Photographs';

-- Get the root path of the FileTable.

DECLARE @FileTableRoot varchar(1000);

SELECT @FileTableRoot = FileTableRootPath('dbo.PhotoTable');

-- Update the PhotoMetadata table.

-- Replace the File Server UNC path with the FileTable path.

UPDATE PhotoMetadata

SET UNCPath = REPLACE(UNCPath, @UNCPathRoot, @FileTableRoot);

-- Update the pathlocator column to contain the pathlocator IDs from the FileTable.

UPDATE PhotoMetadata

SET pathlocator = GetPathLocator(UNCPath);

**FileTable Functions, Stored Procedures, and Views**

**Functions**

| Object | Status | More Information |
| --- | --- | --- |
| [FileTableRootPath](https://docs.microsoft.com/en-us/sql/relational-databases/system-functions/filetablerootpath-transact-sql) | **Added** | [Work with Directories and Paths in FileTables](https://docs.microsoft.com/en-us/sql/relational-databases/blob/work-with-directories-and-paths-in-filetables) |
| [GetFileNamespacePath](https://docs.microsoft.com/en-us/sql/relational-databases/system-functions/getfilenamespacepath-transact-sql) | **Added** | [Work with Directories and Paths in FileTables](https://docs.microsoft.com/en-us/sql/relational-databases/blob/work-with-directories-and-paths-in-filetables) |
| [GetPathLocator](https://docs.microsoft.com/en-us/sql/relational-databases/system-functions/getpathlocator-transact-sql) | **Added** | [Work with Directories and Paths in FileTables](https://docs.microsoft.com/en-us/sql/relational-databases/blob/work-with-directories-and-paths-in-filetables) |

**Stored Procedures**

| Object | Status | More Information |
| --- | --- | --- |
| [sp\_kill\_filestream\_non\_transacted\_handles](https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/filestream-and-filetable-sp-kill-filestream-non-transacted-handles) | **Added** | [Manage FileTables](https://docs.microsoft.com/en-us/sql/relational-databases/blob/manage-filetables) |

**Catalog Views**

| Object | Status | More Information |
| --- | --- | --- |
| [sys.database\_filestream\_options](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-database-filestream-options-transact-sql) | **Added** | [Enable the Prerequisites for FileTable](https://docs.microsoft.com/en-us/sql/relational-databases/blob/enable-the-prerequisites-for-filetable) |
| [sys.filetable\_system\_defined\_objects](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-filetable-system-defined-objects-transact-sql) | **Added** | [Create, Alter, and Drop FileTables](https://docs.microsoft.com/en-us/sql/relational-databases/blob/create-alter-and-drop-filetables) [Manage FileTables](https://docs.microsoft.com/en-us/sql/relational-databases/blob/manage-filetables) |
| [sys.filetables](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-filetables-transact-sql) | **Added** | [Manage FileTables](https://docs.microsoft.com/en-us/sql/relational-databases/blob/manage-filetables) |

**Dynamic Management Views**

| Object | Status | More Information |
| --- | --- | --- |
| [sys.dm\_filestream\_non\_transacted\_handles](https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-filestream-non-transacted-handles-transact-sql) | **Added** | [Manage FileTables](https://docs.microsoft.com/en-us/sql/relational-databases/blob/manage-filetables) |