Contents

[TABLE TYPES IN SQL SERVER 1](#_Toc165199668)

[Types of SQL tables 2](#_Toc165199669)

[System tables 2](#_Toc165199670)

[**Permanent or user-defined table** 2](#_Toc165199671)

[**Temporary tables** 3](#_Toc165199672)

[External tables 4](#_Toc165199673)

[Graph tables 4](#_Toc165199674)

[FILE TABLE EXAMPLE 6](#_Toc165199675)

[LEDGER EXAMPLE 14](#_Toc165199676)

[Temporal tables 21](#_Toc165199677)

[**Demo** 23](#_Toc165199678)

[REFERENCE URL 31](#_Toc165199679)

# TABLE TYPES IN SQL SERVER

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DIFFERENT TYPES OF TABLE | | | | | | | |  |
| TABLE | TEMP | MEMORY OPTIMZED | TEMPORAL | LEDGER | GRAPH | EXTERNAL | FILETABLE | COMPRESSED TABLE |

|  |  |
| --- | --- |
| **TABLE TYPES** | |
| TABLE(User Defined Table /System Defined Table) | Users can define their own table structure, columns, data types, constraints and indexes as per their application requirements. These tables are called user-defined tables.  These tables are always stored in the database unless someone explicitly drops them. Therefore, these are also known as permanent tables. |
| Temp Tables | we need to store data temporarily in the database for calculation, manipulation or storing intermediate results |
| MEMORY OPTIMIZED TABLE | Stores the data in-memory of the database.  The main benefit of memory-optimized tables are that rows in the table are read from and written to memory which results in non-blocking transactions at super-fast speed. The second copy of the data is stored on the disk and during database recovery, data is read from the disk-based table. |
| TEMPORAL TABLE | Temporal tables, also known as system-versioned tables, provide us with new functionality to track data changes. It allows SQL Server to maintain and manage the history of the data in the table automatically. |
| LEDGER TABLE | Ledger helps streamline audits, providing cryptographic proof of data integrity to auditors. |
| GRAPH TABLE | Graph use a collection of different nodes (vertices) and edges |
| EXTERNAL TABLE | These tables reference other data sources such as Azure blob storage, Hadoop, Oracle, Excel, ODBC, Bigdata, MongoDB and Teradata using the PolyBase feature of SQL Server |
| FILE TABLE | This table will refer to a directory and will information about the files present in the directory. The FileTable has a fixed schema, which is mostly metadata about the files that you put in it. |
| COMPRESSED TABLE | This table will store the data of the columns of the table in compressed format. The data will be compressed and which will take less memory storage space as compared to the normal table. |

# Types of SQL tables

SQL Server tables can be divided into the following categories.

# System tables

SQL Server stores instance configuration and database properties in a special set of tables. These tables are known as system tables. Users are not permitted to make changes to these tables directly. SQL Server does not allow specific system tables to be queried directly. Instead, it provides system stored procedures, functions, SQL Server Management Objects and Replication Management Objects for querying these components.

# **Permanent or user-defined table**

Users can define their own table structure, columns, data types, constraints and indexes as per their application requirements. These tables are called user-defined tables.

These tables are always stored in the database unless someone explicitly drops them. Therefore, these are also known as permanent tables.

There are multiple ways to create a user-defined table in SQL Server.

* SQL Server Management Studio GUI
* Using T-SQL script

–Permanent Table  
Create table TableA  
(  
ID int,  
[Name] varchar(50)  
)

# **Temporary tables**

Sometimes, we need to store data temporarily in the database for calculation, manipulation or storing intermediate results. In these cases, we can utilize temporary tables that are always kept in the TempDB system database.

SQL Server has two types of temporary tables:

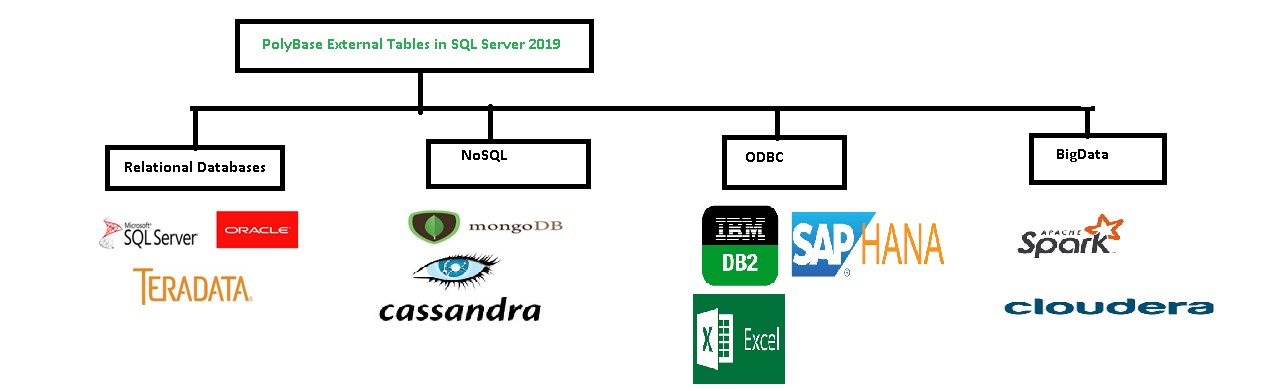
* **Local:** Each local temporary table starts with a sign (#). Its scope is limited to the current connection. SQL Server automatically drops these tables once the user disconnects.
* **Global:** Each global temporary table starts with a sign (##). All users can reference the global temporary tables. If all users that are referencing the global table are connected, SQL Server drops it.

–TEMPORARY Table ( Local )  
Create table #TableA  
(  
ID int,  
[Name] varchar(50)  
)

–Global Temporary Table  
Create table ##TableB  
(  
ID int,  
[Name] varchar(50)  
)

# External tables

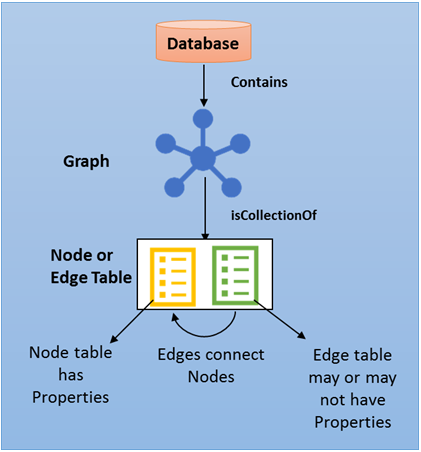
External tables are a particular type of table that can be used in SQL Server 2016 onward. These tables reference other data sources such as Azure blob storage, Hadoop, Oracle, Excel, ODBC, Bigdata, MongoDB and Teradata using the PolyBase feature of SQL Server.



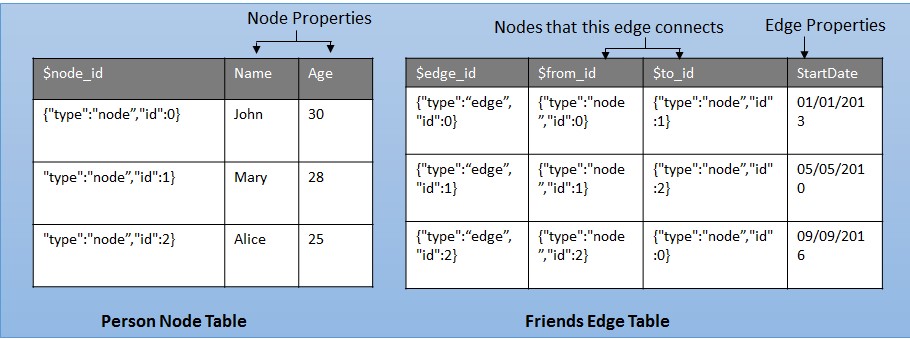
You can refer to this [documentation](https://www.sqlshack.com/category/technologies/polybase/) to further explore PolyBase in SQL Server.

# Graph tables

[SQL Server graph databases](https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-architecture?view=sql-server-ver15) use a collection of different nodes (vertices) and edges (relationships).



* **Node table**: The node table is a collection of nodes having similar types. For example, the person node table has all person nodes in a graph.
* **Edge table**: The edge table is a collection of similar edges. For example, a friend table holds all edges that connect a person to another person.



# FILE TABLE EXAMPLE

With **SQL Server 2012**, a new table type **FileTable** is introduced for SQL Server developers and administrators. FileTable is a feature developed based on the FileStream feature that was introduced with Microsoft SQL Server 2008.

To **create FileTable** is easy but has some prerequisites. Before completing these prerequisites if t-sql developers try to **create FileTable** using following CREATE TABLE command:

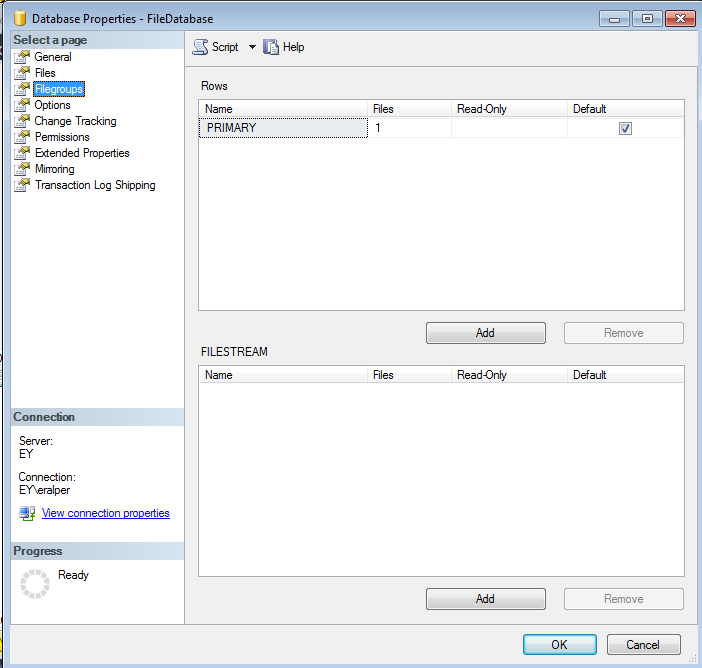
CREATE TABLE ImagesTable AS FileTable  
-- or  
CREATE TABLE ImagesTable AS FileTable  
WITH (  
 FileTable\_Directory = 'ImagesDirectory',  
 FileTable\_Collate\_Filename = database\_default  
)

Code

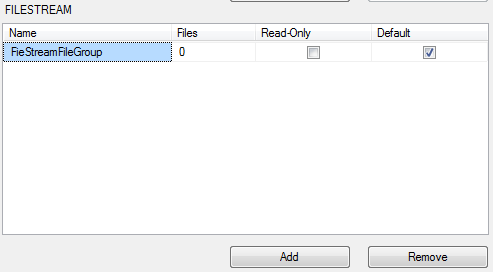
The following SQL Engine error might happen

**Msg 1969, Level 16, State 1, Line 1**  
**Default FILESTREAM filegroup is not available in database 'DatabaseName'**

If you open database properties and display FileGroups tab you will see no filegroup is created or no filegroup is defined as default FileStream filegroup. As you can see in the below screenshot, in the lower section of the screen which is titled as "**FILESTREAM**" there is no filegroup defined.



You can create a new filegroup in the FILESTREAM section and mark it as default

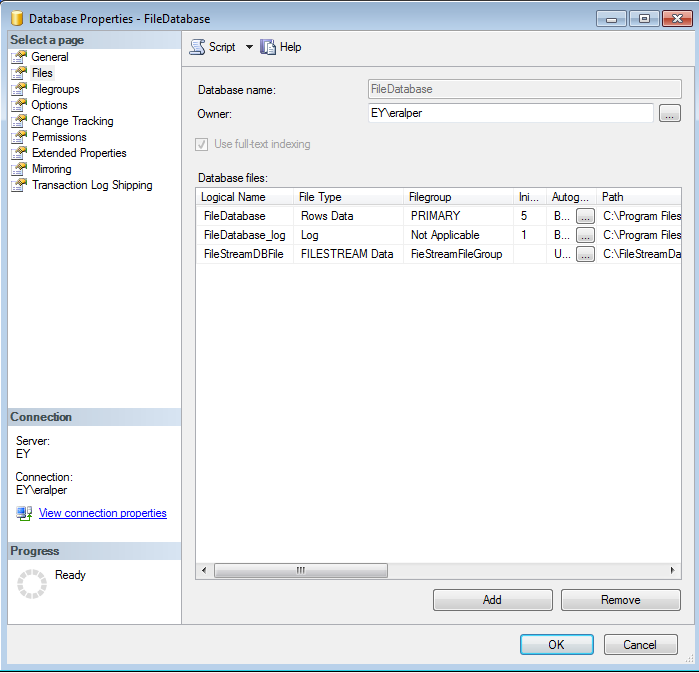


After filegroup is created, it is time to add a new database file with file type is FileStream Data. The default filestream filegroup will be automatically assigned.

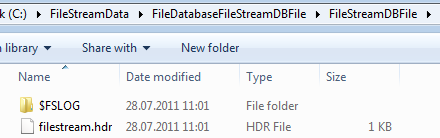
You can change filegroup assignment to a non-default filegroup if you wish from the displayed combobox.

One last step is configuring the Path for the database file.

This will be a physical file folder where the FileStream database file will be stored.



After database file is created, the contents of the filestream database file folder be as follows:

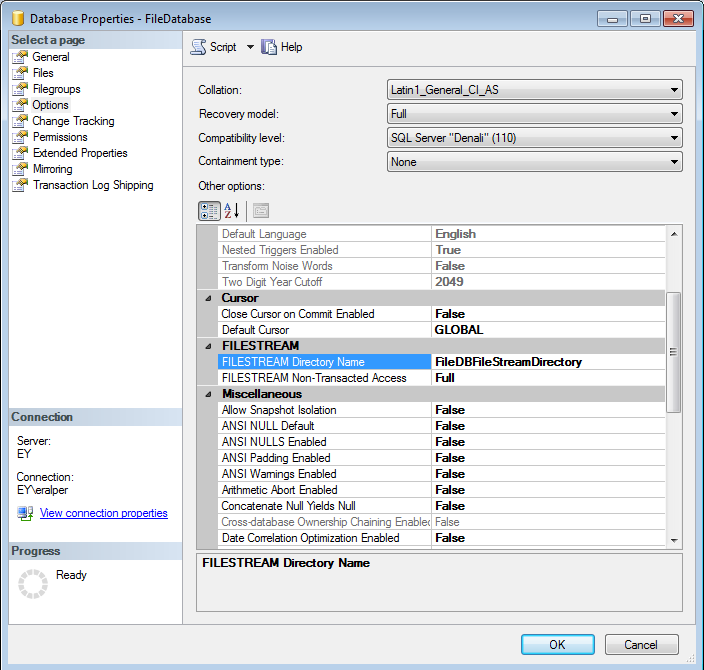


At this step, if you try to create FileTable using "CREATE TABLE ImagesTable AS FileTable" sql command, you will get the following SQL exception

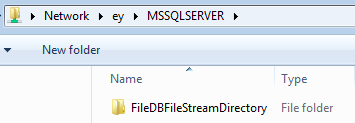
**Msg 33414, Level 16, State 1, Line 2**  
**FileTable objects require the FILESTREAM database option DIRECTORY\_NAME to be non-NULL. To create a FileTable in the database 'FileDatabase', set the DIRECTORY\_NAME option to a non-NULL value using ALTER DATABASE. Or, to set the DIRECTORY\_NAME option to NULL, in the database 'FileDatabase' disable or drop the existing FileTables.**

As the error indicates, the database option **Directory Name** has not been set yet. **Directory Name** can be configured via SQL Server Management Studio or using ALTER DATABASE command.

If you use SSMS, open database properties window and switch to database options tab



Now a new directory is created for the database FileStream data in the SQL Server FileStream network share



Now we can create a FileTable in SQL Server 2012 database

CREATE TABLE ImagesTable AS FileTable  
WITH (  
 FileTable\_Directory = 'ImagesDirectory',  
 FileTable\_Collate\_Filename = database\_default  
)

Code

Now you can copy and paste images directly into FileTable directory

You can even execute a SQL SELECT statement over the new FileTable ImagesTable in order to see the manually copied image file is stored within the FileTable

SELECT \* FROM ImagesTable

Code

How do you create a filetable? I assume you’ve [enabled Filestream](http://voiceofthedba.wordpress.com/2012/03/07/enabling-filestream-in-sql-server-2012/) and [created a filegroup for your filestream and filetable data](http://voiceofthedba.wordpress.com/2012/05/14/create-a-filestream-filegroup-for-filetables-sql-server-2012/). Then you just do this:

-- Create a filetable

CREATE TABLE AuthorDrafts

AS FileTable

GO

The only optional part of this statement is the table name. No other options, no columns, no schema needed. The FileTable has a fixed schema, which is mostly metadata about the files that you put in it.

If I were to select from this table, I’d use this statement. I’m not showing all the columns in the results since there are a lot, but they are in the select.

-- check the table.

select

stream\_id , file\_stream , name , path\_locator ,

parent\_path\_locator ,

file\_type ,

cached\_file\_size ,

creation\_time ,

last\_write\_time ,

last\_access\_time ,

is\_directory ,

is\_offline ,

is\_hidden ,

is\_readonly ,

is\_archive ,

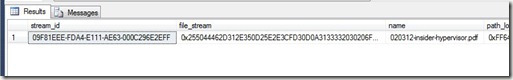
is\_system ,

is\_temporary

from AuthorDrafts;

go

Most of these are really meta data about the file. If I were to drop a table in the share, I’d see results like this:

[](http://voiceofthedba.files.wordpress.com/2012/05/filetable1.jpg)

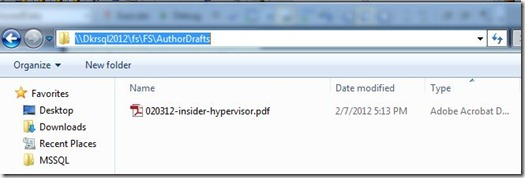
Putting files inside the table is really a drag and drop from Windows. I can get the share name for my filetable from :

-- check the share

select FileTableRootPath('dbo.AuthorDrafts');

go

If I paste this in Explorer, I see my file:

[](http://voiceofthedba.files.wordpress.com/2012/05/filetable2.jpg)

I can drag and drop, or use any scripting commands (Powershell, VBScript, etc) to move files in and out of this share, and they will appear in my table.

It’s that easy to start working with FileTables. How you use them in your application? That’s a whole other series of posts. I’ll work on a few examples you can use over time.

# LEDGER EXAMPLE

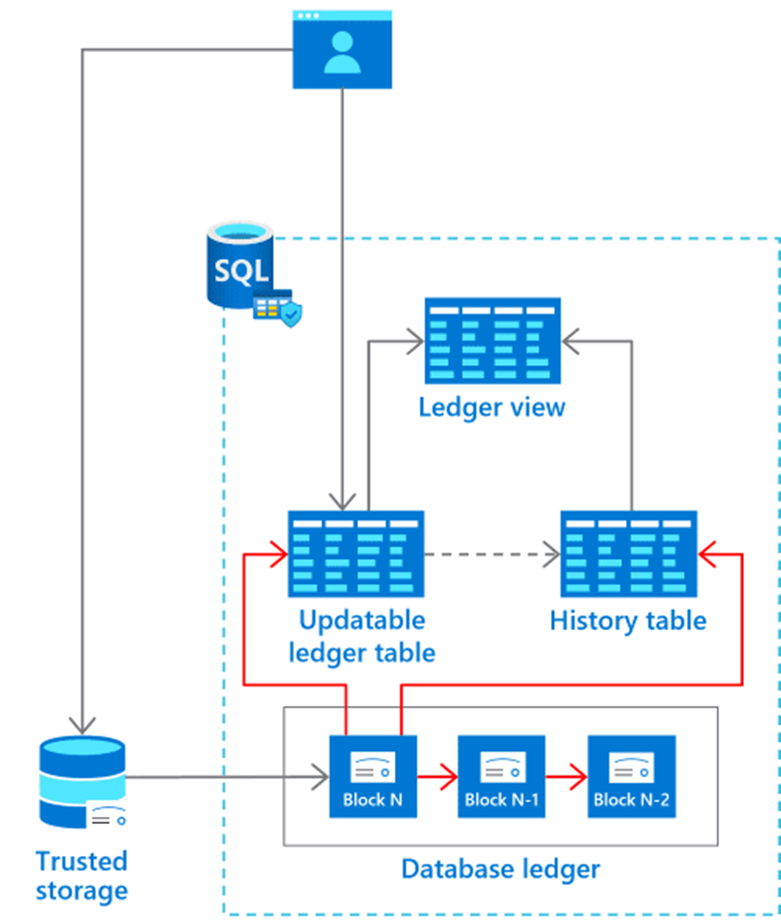
You probably hear or view that on the security point of view, the next version of SQL Server: SQL Server 2022, has a feature named **Ledger**.

It’s already used in Azure but what is Ledger?

In 2 points, I will say:

* Ledger helps streamline audits, providing cryptographic proof of data integrity to auditors.
* Ledger is also a protection of the data from any attacker or high-privileged user, including database or system administrators.

Microsoft provides us a good schema to explain how it’s working:



If any changes is done (DELETE,UPDATE) in your table, the previous value goes in a history table.

This is like Temporal Table, isn’t it? Have a look on [here](https://www.dbi-services.com/blog/sql-server-temporal-table-how-to-store-a-history-table-in-another-file/) (I know the author 😉 )

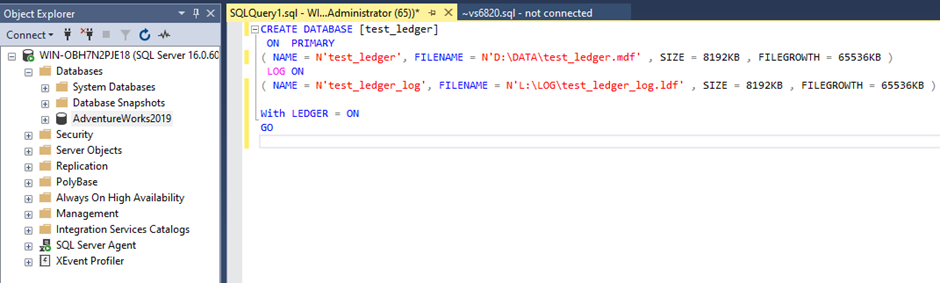
Yes and No. Yes for the principle but no because it’s also protected and going deeper for where does what…

The technology used is the blockchain. Each records will be crypted and have the previous block crypted and a timestamp.  
I this blog, I will just explain how it’s working without the crypted side.

Let’s start our first steps…

We can create directly a database with the option [WITH LEDGER = ON](https://docs.microsoft.com/en-us/sql/t-sql/statements/create-database-transact-sql?view=sql-server-ver16&tabs=sqlpool)

For example:



CREATE DATABASE [test\_ledger]

ON PRIMARY

( NAME = N'test\_ledger', FILENAME = N'D:\DATA\test\_ledger.mdf' , SIZE = 8192KB , FILEGROWTH = 65536KB )

LOG ON

( NAME = N'test\_ledger\_log', FILENAME = N'L:\LOG\test\_ledger\_log.ldf' , SIZE = 8192KB , FILEGROWTH = 65536KB )

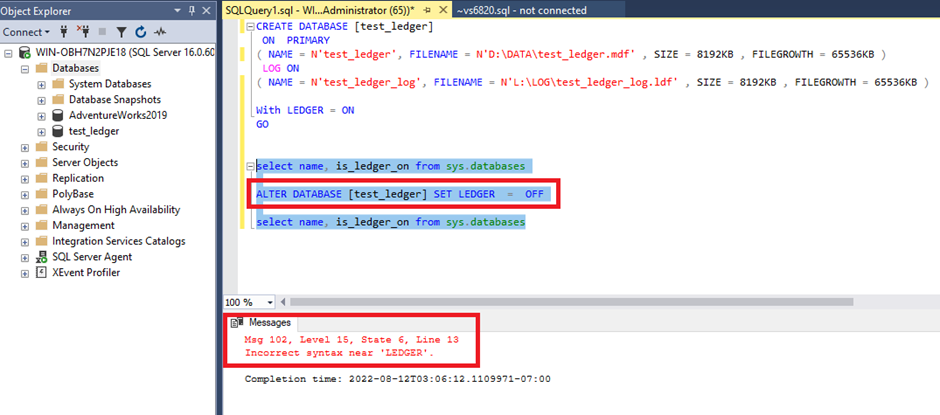
With LEDGER = ON

GO

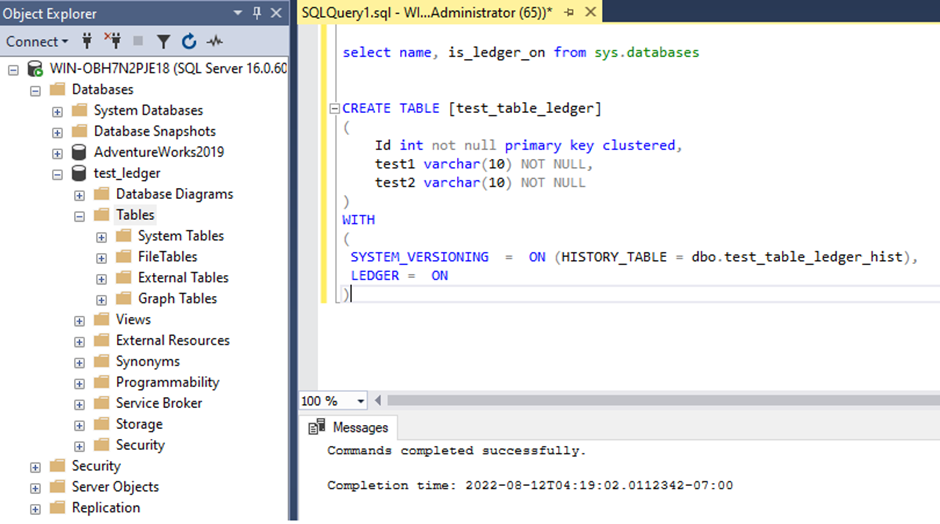
You can easily see with sys.databases, if it’s enable or not with the column name is\_ledger\_on:

select name, is\_ledger\_on from sys.databases

I try also to do an alter database to disable it but it’s not working, I would say “not yet”…



After that, I create a table with the syntax of the temporal table and adding the ledger syntax:



CREATE TABLE [test\_table\_ledger]

(

Id int not null primary key clustered,

test1 varchar(10) NOT NULL,

test2 varchar(10) NOT NULL

)

WITH

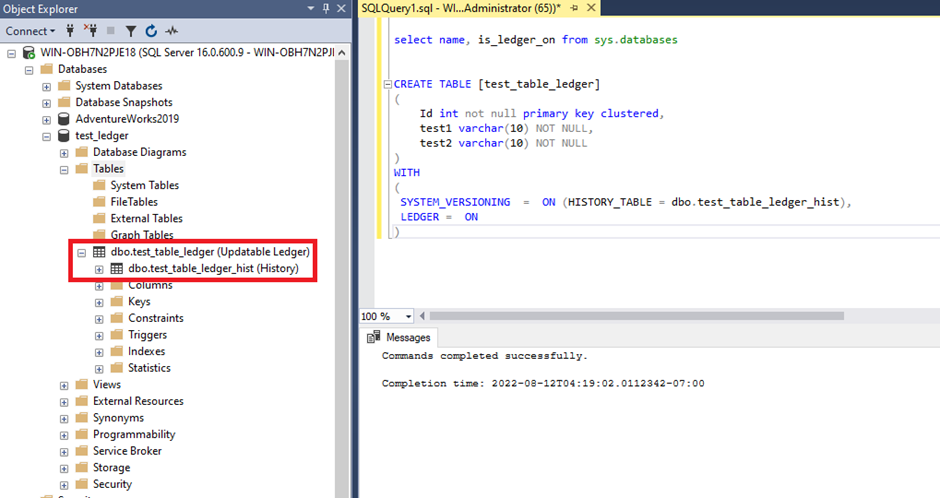
(

SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.test\_table\_ledger\_hist),

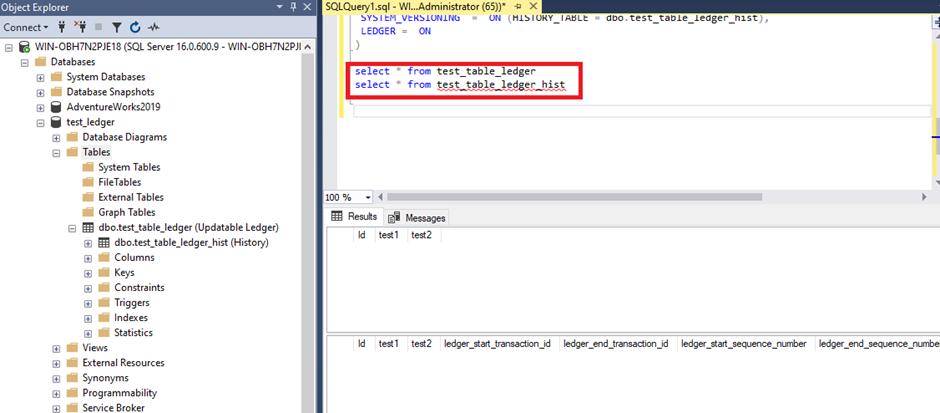
LEDGER = ON

)

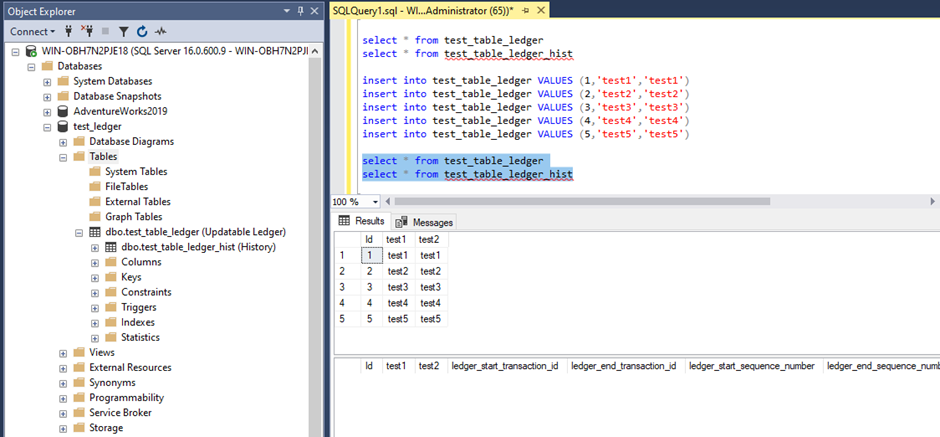
As you can see in the new SSMS, the table is marked as Updatable Ledger and you have an History table associated



If I have a look at these 2 tables, they are empty:

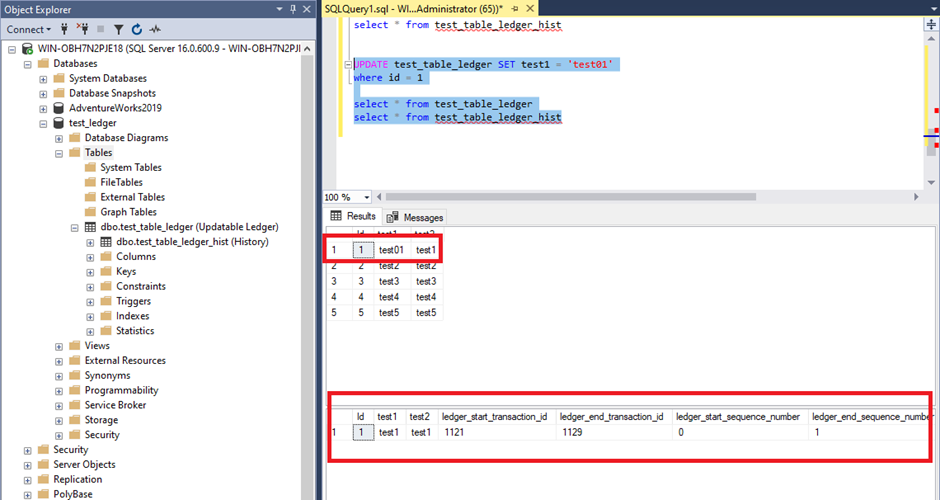


I insert 5 values to have some rows in my table:



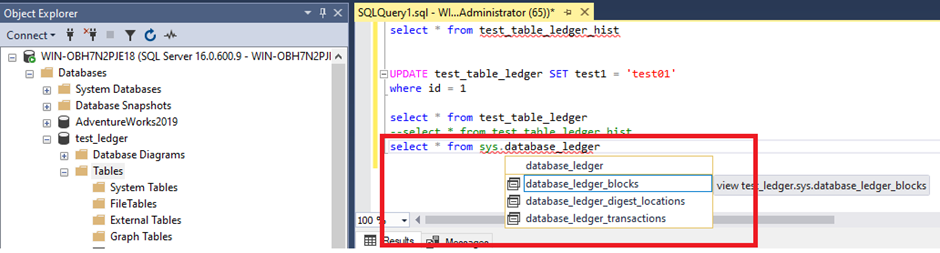
No changes with these inserts, the history table is logically empty.

Now, I will do a simple update of the first row:



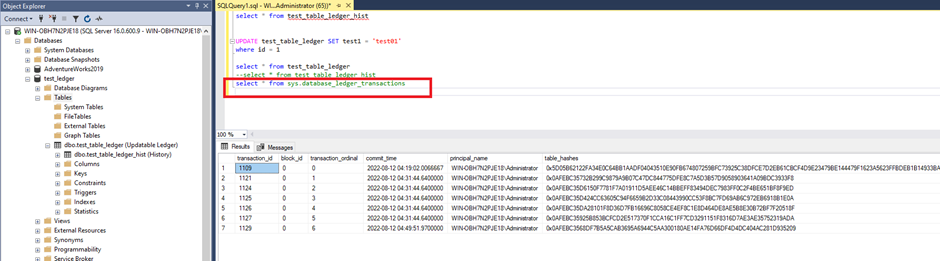
As you can see, the history table has the old value and my table is up to date with the new value

The recommendation of Microsoft is to use the ledger view for the history table and not directly the history table.



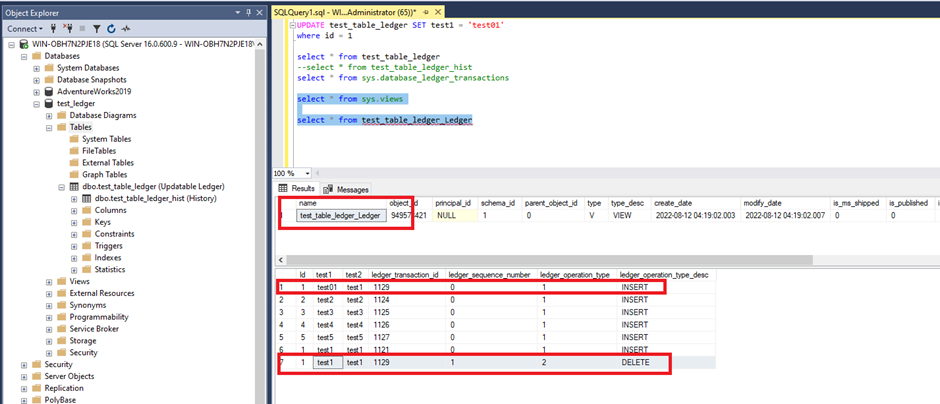
3 views are possible. The first one is [sys.database\_ledger\_transactions](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-database-ledger-transactions-transact-sql?view=sql-server-ver16) and I will take only about this one in this blog

This view give us the history of the database transactions:



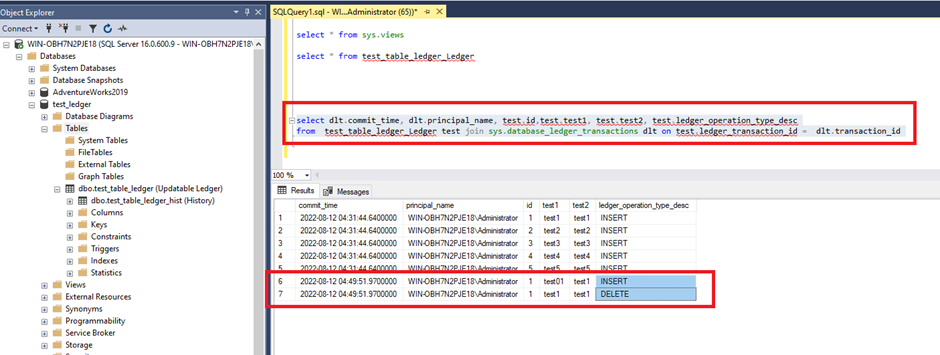
No very digest to read, but if I look, a new view is created though the Ledger with the name of my table + \_Ledger.

In this case you can easily see the operation type



Combining this last view and sys.database\_ledger\_transactions, you have a trace of the operations.

In my case, 5 insert and 1 update (INSERT + DELETE)



select dlt.commit\_time, dlt.principal\_name, test.id,test.test1, test.test2, test.ledger\_operation\_type\_desc

from test\_table\_ledger\_Ledger test join sys.database\_ledger\_transactions dlt on test.ledger\_transaction\_id = dlt.transaction\_id

Another interesting part of this new feature is the stored procedure to copy data from a normal table to a ledger table: [sp\_copy\_data\_in\_batches](https://docs.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-how-to-migrate-data-to-ledger-tables?view=sql-server-ver16)

<https://www.dbi-services.com/blog/sql-server-2022-security-do-you-know-ledger/>

https://www.sqlshack.com/sql-server-filetable-the-next-generation-of-sql-filestream/

# Temporal tables

Temporal tables have been a feature of SQL Server since version 2016. SQL Server professionals have historically had several options to track data change events. The evolution of the data tracking mechanism started with Change tracking (CT), Change Data Capture (CDC) and now Temporal Tables.

**Introduction**

In my experience, I have seen that a few projects still use custom data tracking solutions. And in a few other enterprises, third-party solutions are in place to manage the same. In this article, we’ll discuss temporal tables in SQL Server 2016.

**GDPR (General Data Protection Regulation)**

As we all know, the General Data Protection Regulation (GDPR), a new data privacy regulatory act, introduces significant changes in the way organizations collect or use data. Enterprises must adhere to the data privacy and protection act.

* ***Note:*** *A detailed explanation of GDPR concepts can be found in the article:* [*Using production data for testing in a post GDPR world*](https://www.sqlshack.com/using-production-data-testing-post-gdpr-world/)

To comply with GDPR requirements, organizations strive towards working with various tools and techniques.

SQL Server offers various native tools and techniques and is capable of fulfilling most of the security requirements for being GDPR complaint.

* ***Note****: A detailed explanation of SQL Server Features that support GDPR compliance can be found in the article:* [*SQL Server data security feature RLS (Row-Level Security) and GDPR*](https://www.sqlshack.com/sql-server-data-security-feature-rls-row-level-security-and-gdpr/)

**Getting Started**

What is Temporal Table?

Temporal tables, also known as system-versioned tables, provide us with new functionality to track data changes. It allows SQL Server to maintain and manage the history of the data in the table automatically. This feature provides a full history of every change made to the data.

It was first introduced in ANSI (American National Standards Institute) [SQL 2011 standard](https://en.wikipedia.org/wiki/SQL:2011). Now, it’s available in SQL Server 2016 and later versions.

Temporal tables are considered as one of the critical features used to audit SQL Server data.

The two period columns which are of datetime2 data type, **SysStartTime** and **SysEndTime** define the current and historical data for every row of the table. It defines the validity of the data. The DateTime range has a value that is appropriate during that timeframe. By querying the table, we can easily see the transitioning of the data to its various states within the specific date-time intervals.

Why Temporal tables?

The following are some usage scenarios of Temporal tables

1. Auditing
2. Rebuilding the data in case of inadvertent changes
3. Projecting and reporting for historical trend analysis
4. Protecting the data in case of accidental data loss

**Demo**

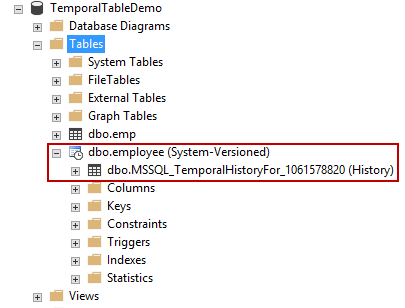
Let’s jump in to get started with the demonstration.

The following example creates an *employee* temporal table created with a few sample rows.

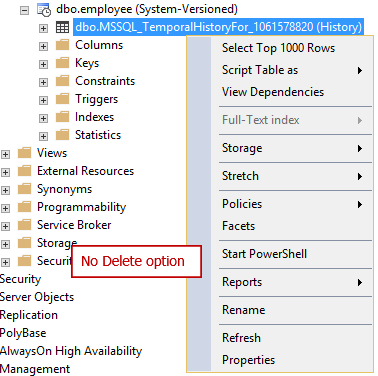
|  |
| --- |
|  |
| CREATE TABLE dbo.employee(    EMPNO   INT,    ENAME    VARCHAR(10),    JOB      VARCHAR(9),    MGR      INT,    HIREDATE DATE,    SAL      NUMERIC(7,2),    COMM     NUMERIC(7,2),    DEPTNO   INT,       CONSTRAINT EMP\_PK PRIMARY KEY (EMPNO),     SysStartTime datetime2 GENERATED ALWAYS AS ROW START NOT NULL    ,SysEndTime datetime2 GENERATED ALWAYS AS ROW END NOT NULL    ,PERIOD FOR SYSTEM\_TIME (SysStartTime,SysEndTime)) WITH (SYSTEM\_VERSIONING = ON); | |

By default, the SQL Server will automatically generate a name for the history table in the following format

dbo.MSSQL\_TemporalHistoryFor\_<ObjectID>.



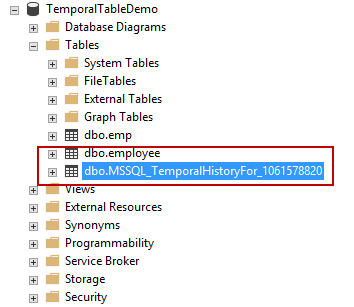
If you browse the table, we can see the absence of “Delete” option for Temporal Tables properties.



To get the delete option, first, turn off the **system\_versioning** setting using the **alter table** statement. After that, you’ll notice that the tables, temporal table and history table become a regular table.

**You can also use the SSMS generate script option to drop the temporal table.**

|  |
| --- |
|  |
| **ALTER TABLE [dbo].employee SET ( SYSTEM\_VERSIONING = OFF )**  **GO** | |



Now, the tables can be dropped either by using T-SQL or SSMS.

|  |
| --- |
|  |
| DROP TABLE [dbo].employee  GO  DROP TABLE [dbo].MSSQL\_TemporalHistoryFor\_1061578820 | |

|  |
| --- |
| ALTER TABLE [dbo].employee SET ( SYSTEM\_VERSIONING = OFF )  GO  ALTER TABLE [dbo].employee SET ( SYSTEM\_VERSIONING = on )  GO  ALTER TABLE [dbo].employee  set (SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.EmployeeHist, DATA\_CONSISTENCY\_CHECK = ON)); |

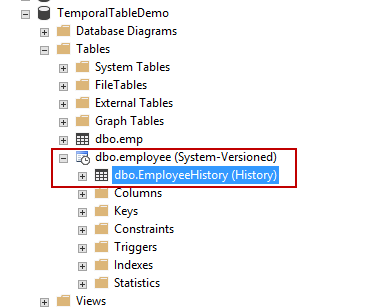
How to Create a Custom Name for Temporal History table

In this section, we’ll see how to create a custom temporal table along with a consistency checker option. In this case, the *employeeHistory* is the history table and **DATA\_CONSISTENCY\_CHECK=ON** is used with the WITH clause of the create table statement.

The create table statement is discussed in detail [here](https://docs.microsoft.com/en-us/sql/t-sql/statements/create-table-transact-sql).

|  |
| --- |
|  |
| CREATE TABLE dbo.employee(    EMPNO   INT,    ENAME    VARCHAR(10),    JOB      VARCHAR(9),    MGR      INT,    HIREDATE DATE,    SAL      NUMERIC(7,2),    COMM     NUMERIC(7,2),    DEPTNO   INT,       CONSTRAINT EMP\_PK PRIMARY KEY (EMPNO),     SysStartTime datetime2 GENERATED ALWAYS AS ROW START NOT NULL    ,SysEndTime datetime2 GENERATED ALWAYS AS ROW END NOT NULL    ,PERIOD FOR SYSTEM\_TIME (SysStartTime,SysEndTime)) WITH (SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.EmployeeHistory, DATA\_CONSISTENCY\_CHECK = ON)); | |

In the following output, we can see that the custom history table *dbo.EmployeeHistory* is created as part of the Create table statement.

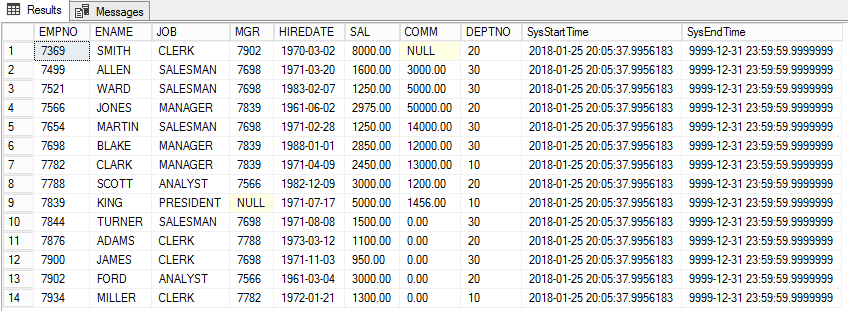


Insert a few records into the employee table

|  |
| --- |
|  |
| INSERT INTO employee (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)values  (7369, 'SMITH', 'CLERK', 7902, '02-MAR-1970', 8000, NULL, 20),  (7499, 'ALLEN', 'SALESMAN', 7698, '20-MAR-1971', 1600, 3000, 30),  (7521, 'WARD', 'SALESMAN', 7698, '07-FEB-1983', 1250, 5000, 30),  (7566, 'JONES', 'MANAGER', 7839, '02-JUN-1961', 2975, 50000, 20),  (7654, 'MARTIN', 'SALESMAN', 7698, '28-FEB-1971', 1250, 14000, 30),  (7698, 'BLAKE', 'MANAGER', 7839, '01-JAN-1988', 2850, 12000, 30),  (7782, 'CLARK', 'MANAGER', 7839, '09-APR-1971', 2450, 13000, 10),  (7788, 'SCOTT', 'ANALYST', 7566, '09-DEC-1982', 3000, 1200, 20),  (7839, 'KING', 'PRESIDENT', NULL, '17-JUL-1971', 5000, 1456, 10),  (7844, 'TURNER', 'SALESMAN', 7698, '08-AUG-1971', 1500, 0, 30),  (7876, 'ADAMS', 'CLERK', 7788, '12-MAR-1973', 1100, 0, 20),  (7900, 'JAMES', 'CLERK', 7698, '03-NOV-1971', 950, 0, 30),  (7902, 'FORD', 'ANALYST', 7566, '04-MAR-1961', 3000, 0, 20),  (7934, 'MILLER', 'CLERK', 7782, '21-JAN-1972', 1300, 0, 10) | |

Query the employee table

|  |
| --- |
|  |
| select \* from employee; | |

[](https://www.sqlshack.com/wp-content/uploads/2019/02/our-initial-test-table-for-creating-a-temporal-tab.png)

Now, If you want to hide the **SysStartTime** and **SysEndTime** columns from viewing, add a **hidden** keyword in the *create table* statement so that the column does not show up in a select query.

|  |
| --- |
|  |
| CREATE TABLE dbo.employee(    EMPNO   INT,    ENAME    VARCHAR(10),    JOB      VARCHAR(9),    MGR      INT,    HIREDATE DATE,    SAL      NUMERIC(7,2),    COMM     NUMERIC(7,2),    DEPTNO   INT,       CONSTRAINT EMP\_PK PRIMARY KEY (EMPNO),     SysStartTime datetime2 GENERATED ALWAYS AS ROW START HIDDEN NOT NULL    ,SysEndTime datetime2 GENERATED ALWAYS AS ROW END HIDDEN NOT NULL    ,PERIOD FOR SYSTEM\_TIME (SysStartTime,SysEndTime)) WITH (SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.EmployeeHistory, DATA\_CONSISTENCY\_CHECK = ON));   |  | | --- | | SysStartTime datetime2 GENERATED ALWAYS AS ROW START NOT NULL    ,SysEndTime datetime2 GENERATED ALWAYS AS ROW END NOT NULL    ,PERIOD FOR SYSTEM\_TIME (SysStartTime,SysEndTime)) WITH (SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.EmployeeHistory, DATA\_CONSISTENCY\_CHECK = ON)); | | |

Let us perform SQL Update DML operations to understand the usage of SYSTEM\_TIME column.

For example, SMITH’s salary has got an increment of 2000. The update SQL is as follows.

|  |
| --- |
|  |
| UPDATE EMPLOYEE  SET SAL=SAL+2000  WHERE EMPNO=7369    SELECT \* FROM EMPLOYEE WHERE EMPNO=7369 | |

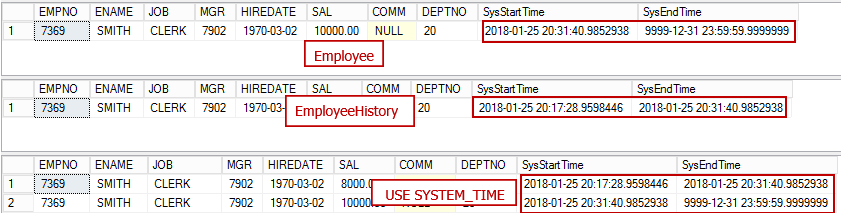


After the update, we can see that the Salary column is updated. The column SysEndTime is a reference pointer to indicate the active record of the table. It’s really easy to audit the data flow by referring the SysStartTime and SysEndTime.

|  |
| --- |
|  |
| SELECT \*, SysStartTime,SysEndTime FROM EMPLOYEE WHERE EMPNO=7369    SELECT \* FROM EmployeeHistory WHERE EMPNO=7369    SELECT \*, SysStartTime,SysEndTime FROM EMPLOYEE  FOR SYSTEM\_TIME ALL  order by empno, SysEndTime | |

The SysEndTime ‘9999-12-31 23:59:59.9999999’ always refers to the active record set. The *dbo.emplyeehistory* table has the previous reference pointers of the modified data. In this case, the column, **salary**, is seen as updated on “2018-01-25 20:31:40.9852938”. The unmodified record made an entry into the history table with the updated timestamp. The updated timestamp becomes the SysStartTime column of the modified record.

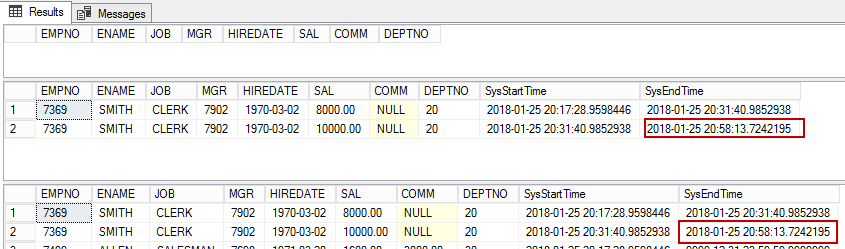
The entire update process can be easily traced by referring the Sys columns that is arranged in a Zig-Zag manner.



Now, Let us perform the SQL Delete operation to delete the employee record 7369 and see what happens

|  |
| --- |
|  |
| DELETE from EMPLOYEE WHERE EMPNO=7369    SELECT \* FROM Employee WHERE EMPNO=7369    SELECT \* FROM EmployeeHistory WHERE EMPNO=7369    SELECT \*, SysStartTime,SysEndTime FROM EMPLOYEE  FOR SYSTEM\_TIME ALL  order by empno, SysEndTime | |

The output is an empty result set. The SysEndTime is updated with the timestamp of DELETE statement. This indicates that the record was deleted from the table.

>

The FOR SYSTEM\_TIME clause has many variations and options. It is further classified into four temporal sub-clauses. This provides a way to query the data across current and history tables.

1. AS OF <datetime>
2. FROM <startdatetime> TO <enddatetime>
3. BETWEEN <startdatetime> AND <enddatetime>
4. CONTAINED IN (<startdatetime> , <enddatetime>)
5. ALL

The AS OF clause is used when there is a need to rebuild the original state of the data and need to know the state it was at any specific time in the past. This is possible by specifying the date time as its input.

|  |
| --- |
|  |
| SELECT \*, [SysStartTime],[SysEndTime]  FROM [dbo].EMPLOYEE  FOR SYSTEM\_TIME AS OF '2018-01-25 20:17:28.9598446'  WHERE EMPNO=7369 | |

Here are some other sub-clause temporal options that are useful for data audit.

|  |
| --- |
|  |
| SELECT \* ,SysStartTime,SysEndTime FROM EMPLOYEE  FOR SYSTEM\_TIME BETWEEN '2018-01-25 20:17:28.9598446' AND '2018-01-25 20:58:13.7242195';    SELECT \* ,SysStartTime,SysEndTime FROM EMPLOYEE  FOR SYSTEM\_TIME CONTAINED IN('2018-01-25','2018-01-26')    SELECT \*, SysStartTime,SysEndTime FROM EMPLOYEE  FOR SYSTEM\_TIME ALL  order by empno, SysEndTime | |

Summary

In this article, we see the step-by-step procedures to setup temporal tables in SQL Server. Temporal Tables are generally useful in scenarios that require tracking the history of data changes. In some cases, it can be used to reconstruct the data. Its steps are pretty simple and straightforward. It doesn’t require writing triggers or stored procedures or any application code change. Data retrieval is made very simple using the SYSTEM\_TIME clause along with the associated sub-clause.

The history table can grow big. It depends on the number of transactions on the temporal table so, space is the biggest tradeoff. You can consider purging the data from the history table.

|  |
| --- |
| ENAME VARCHAR(10),  JOB VARCHAR(9),  MGR INT,  HIREDATE DATE,  SAL NUMERIC(7,2),  COMM NUMERIC(7,2),  DEPTNO INT,  CONSTRAINT EMP\_PK PRIMARY KEY (EMPNO),  SysStartTime datetime2 GENERATED ALWAYS AS ROW START NOT NULL  ,SysEndTime datetime2 GENERATED ALWAYS AS ROW END NOT NULL  ,PERIOD FOR SYSTEM\_TIME (SysStartTime,SysEndTime)) WITH (SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.EmployeeHistory, DATA\_CONSISTENCY\_CHECK = ON));  ------------  CREATE TABLE dbo.employee(  EMPNO INT,  ENAME VARCHAR(10),  JOB VARCHAR(9),  MGR INT,  HIREDATE DATE,  SAL NUMERIC(7,2),  COMM NUMERIC(7,2),  DEPTNO INT,  CONSTRAINT EMP\_PK PRIMARY KEY (EMPNO),  SysStartTime datetime2 GENERATED ALWAYS AS ROW START NOT NULL  ,SysEndTime datetime2 GENERATED ALWAYS AS ROW END NOT NULL  ,PERIOD FOR SYSTEM\_TIME (SysStartTime,SysEndTime)) WITH (SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.EmployeeHistory, DATA\_CONSISTENCY\_CHECK = ON));  --------------------------  UPDATE EMPLOYEE SET SAL=SAL+2000 WHERE EMPNO=7369    SELECT \* FROM EMPLOYEE WHERE EMPNO=7369  select \* from EmployeeHist  SELECT \* FROM EMPLOYEE WHERE EMPNO=7369  select \* from EmployeeHist |

# REFERENCE URL

<https://www.sqlservercentral.com/blogs/creating-a-filetable>

<https://www.kodyaz.com/t-sql/default-filestream-filegroup-is-not-available-in-database.aspx>

<https://learn.microsoft.com/en-us/sql/relational-databases/blob/enable-and-configure-filestream?view=sql-server-ver16>

https://blog.quest.com/a-beginners-guide-to-sql-tables/

https://www.sqlshack.com/temporal-tables-in-sql-server/