Temporal Tables in SQL Server 2016

SQL Server 2016 has provided with the new temporal table support which tracks the history of all the data changes to a table. With Temporal tables SQL Server is able to store all the older data records into in a history table while keeping the current records in the original table. In this article we will explore using the temporal table feature of SQL Server 2016 to create a history table.

What Is a Temporal Table?

A temporal table is just another SQL Server table that contains the old rows for a corresponding SQL Server table. A temporal table is a new type of table that provides correct information about stored facts at any point in time. It is basically just a history table of old rows. Each temporal table consists of two tables actually, one for the current data and one for the historical data. Every time an existing record is updated, the old row is placed in the associated temporal table automatically. A temporal table can also be called a history table. Using this new feature in SQL Server 2016 means you can now track changes to a table overtime without having to write any application logic. SQL Server will place the older rows in the temporal tables automatically.

SQL Server manages the movement of records between the original table and the temporal history table. The original table and the temporal table contain a set of period columns. The period columns, consist of a begin date and an end date column for the record. These two dates represent the period of time that a record is active, and are defined as datetime2 columns. When a record is updated the SQL Server engine automatically updates the end date on the record being updated to the current UTC time, and then moves the existing record to the temporal table. When your application creates a new record in the normal or original table the period begin date is set to the UTC time based on a default value for the column, and then the end date is set to the default value for the end date column.

Implementation

To better understand how this works let me show you an example.

We have *Users* table in our SQL Server 2016 database. The *Users* table contains user details and the status of each user i.e. active or disabled. We would like to store the old versions of rows in the *Users* table. By keeping older versions of records we will be able to track the user status changes over time.

Current Users Table

In order to demonstrate how to use a temporal table to track the changes to the *Users* table over time, we will first need to create the *Users* table and populate it with some rows of data. To create and populate the *Users* table let's use the following code:

```
CREATE TABLE [dbo].[Users](
	[user_id] [int] IDENTITY(1,1) NOT NULL,
	[login_id] [varchar](50) NOT NULL,
	[full_name] [varchar](50) NOT NULL,
	[disabled] [char](1) NOT NULL,
 CONSTRAINT [PK_Users] PRIMARY KEY CLUSTERED
(
	[user_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]


INSERT INTO [dbo].[Users] VALUES
('Dev6test606','LORI WENGER-HANSEN','N'),
('Dev6test613','DIVYAKANT AGRAWAL','N'),
('Dev6test625','KATHLEEN M LELEVIER','N'),
('Dev6test626','BETTY EMMONS','N')
GO
```

In this code I created a table named *dbo.Users* and then populated it with four different users.

Setting up Temporal Data on *Users* Table

In order to start collecting historical information for our *dbo.Users* table, we will need to alter the table so it will support temporal data. A SQL Server 2016 temporal table requires a table to have a primary key and a couple of date/time columns. The primary key is needed to be able to match records from the *dbo.Users* table and the temporal table. The two date/time columns will be used to determine the period of time for when the record is valid. Therefore the first thing we need to do is alter our *Users* table to meet the temporal data table requirements. To do that let's run the following code:

```
ALTER TABLE [dbo].[Users]
    ADD [BeginDate] datetime2 GENERATED ALWAYS AS ROW START NOT NULL
        DEFAULT SYSUTCDATETIME(),
            [EndDate] datetime2 GENERATED ALWAYS AS ROW END NOT NULL
        DEFAULT CAST('9999-12-31 23:59:59.9999999' AS datetime2),
    PERIOD FOR SYSTEM_TIME (BeginDate,EndDate);
```

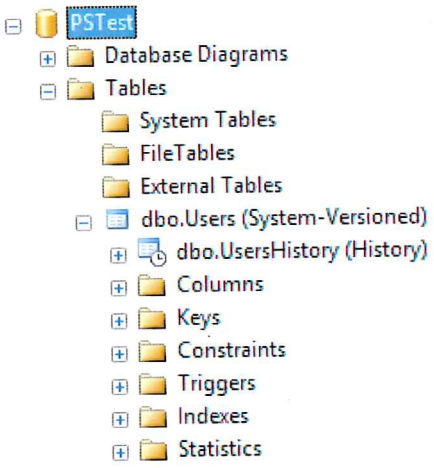
Here you can see that we have created two date columns named *BeginDate* and *EndDate*. These two fields identify a time period when the *Users* table record is valid. Note that we have set the *BeginDate* column value to the current data/time, in UTC format and then EndDate to a date/time that is way into the future. The reason I used UTC is because

support for the temporal table time period is based on UTC time and not the current time zone of the SQL Server instance.

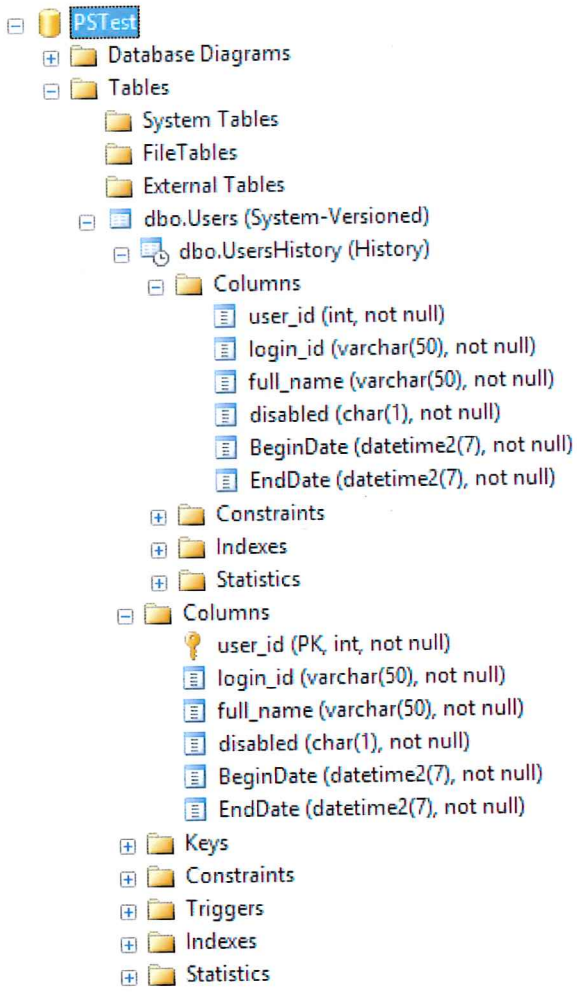
The next step in setting up a temporal table is to identify a history table that goes along with our *dbo.Users* table. To do that let's run the following code:

```
ALTER TABLE [dbo].[Users]
    SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.UsersHistory));
GO
```

Here you can see that the table named dbo.UsersHistory would be our system versioned temporal table. Go to Object Explorer in SQL Server Management Studio (SSMS) to see how we can identify that the dbo.Users table now has a history table associated with it. Below is what we can notice when we browse our database:



Note that our *dbo.Users* table now says it is a "System-Versioned" table. Additionally there is a new node under the *dbo.Users* table that identifies the "History" table *dbo.UsersHistory*. Expand the history table named *dbo.UsersHistory*. Below you can see the definition for the history table in SSMS:



Here you can see that the history table looks exactly like the *dbo.Users* table. At this point we can actually select data from this history table. But since we have yet to update, or delete an existing row from the *dbo.Users* table there are no records in the history table.

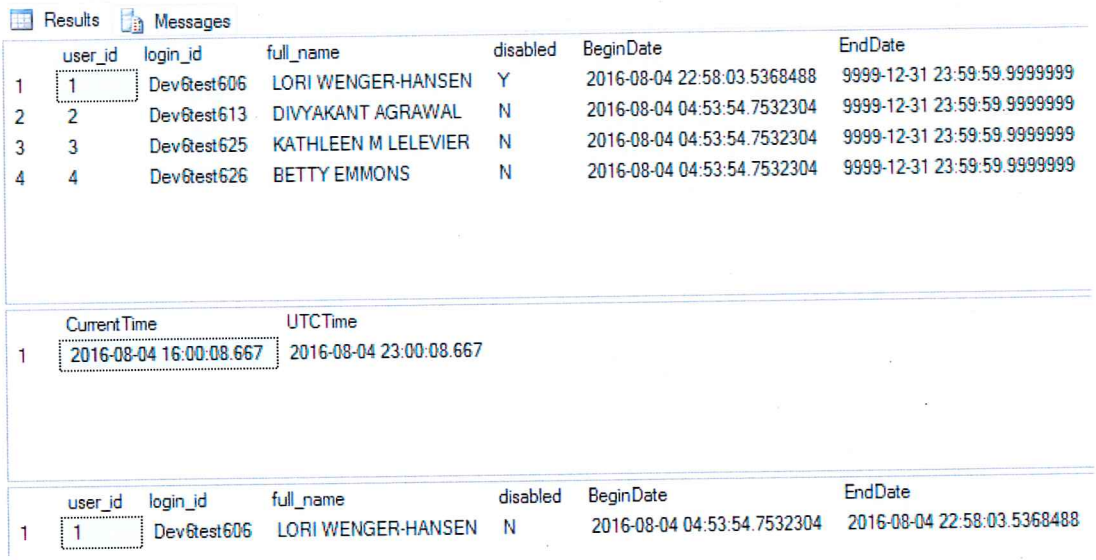
Processing Update Statement against a "System-Versioned" Table

In order to see how a temporal table can hold the history records let us perform an UPDATE statement against *dbo.Users* table. To perform that update we will be using the following script:

```
UPDATE [dbo].[Users]
SET [disabled] = 'Y'
WHERE login_id = 'Dev6test606';
```

When we run the following code we will get the following output:

```
SELECT * FROM [dbo].[Users];
SELECT GETDATE() CurrentTime, GETUTCDATE() UTCTime;
SELECT * FROM [dbo].[UsersHistory];
```



Results Messages

	user_id	login_id	full_name	disabled	BeginDate	EndDate
1	1	Dev6test606	LORI WENGER-HANSEN	Y	2016-08-04 22:58:03.5368488	9999-12-31 23:59:59.9999999
2	2	Dev6test613	DIVYAKANT AGRAWAL	N	2016-08-04 04:53:54.7532304	9999-12-31 23:59:59.9999999
3	3	Dev6test625	KATHLEEN M LELEVIER	N	2016-08-04 04:53:54.7532304	9999-12-31 23:59:59.9999999
4	4	Dev6test626	BETTY EMMONS	N	2016-08-04 04:53:54.7532304	9999-12-31 23:59:59.9999999

	CurrentTime	UTCTime
1	2016-08-04 16:00:08.667	2016-08-04 23:00:08.667

	user_id	login_id	full_name	disabled	BeginDate	EndDate
1	1	Dev6test606	LORI WENGER-HANSEN	N	2016-08-04 04:53:54.7532304	2016-08-04 22:58:03.5368488

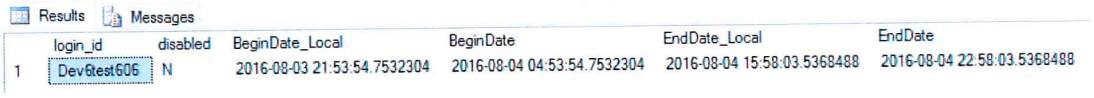
If you look at the code above you can see that we first displayed all the records in the *dbo.Users* table. Here you can see the updated *disabled* field value. We then displayed the local and UTC time, followed by the data from the temporal data table *dbo.UsersHistory*. The data displayed from the temporal table was the old user record prior to updating it. As you can see SQL Server automatically set the *EndDate* on this record to the current UTC date. Remember this is not the local time on our SQL Server machine. This is because temporal data uses UTC dates when calculating the end date of a record. If you compare the *UTCTime* column that we displayed with the *EndDate you* can see they are close to the same time, whereas the *CurrentTime* column is very different, and represents the local time on our machine.

Displaying Period Values in Local Time Format

Remember the begin date and end dates for our temporal tables are updated with the UTC time, and not the local time zone of the SQL Server machine. When looking at date ranges for temporal tables it might be nice to be able to display the period begin and end date in local time. To accomplish this let's run the following code:

```
SELECT login_id, [disabled],
       DATEADD(mi, DATEDIFF(mi, GETUTCDATE(), GETDATE()), BeginDate)
       AS BeginDate_Local,
    BeginDate ,
       DATEADD(mi, DATEDIFF(mi, GETUTCDATE(), GETDATE()), EndDate)
       AS EndDate_Local,
    EndDate
FROM  [dbo].[UsersHistory];
```

When we run this code we will get the following output:



Results | Messages

	login_id	disabled	BeginDate_Local	BeginDate	EndDate_Local	EndDate
1	Dev6test606	N	2016-08-03 21:53:54.7532304	2016-08-04 04:53:54.7532304	2016-08-04 15:58:03.5368488	2016-08-04 22:58:03.5368488

If you review the output above you will see that the *BeginDate_Local* and the *EndDate_Local* will represent the local time on our machines, whereas the *BeginDate* and *EndDate* columns contain the UTC time.

Limitations of Temporal Tables

Remember this is the first version of temporal table support. Like any new version feature there are a number of limitations for temporal tables. This is a partial list of some of those limitations:

- History tables need to be created on the same database as the table that is being versioned.
- You are not able to truncate the history table.
- You are not allowed to modify the rows of data in the history table with an INSERT, UPDATE, and DELETE statement.
- History tables cannot have a primary key, foreign key, or column constraints.

Track Your Data Table Changes with Temporal Tables

Having a historical temporal data table that is automatically populated with data is a great feature. This keeps you from having to write the code for the period date range values. You are able to track the changes of individual rows over time by having temporal tables. Having history records in the temporal table allows you to historically determine what a record looked like for any given time period in the past. If you are looking for adding a history table in the future, consider whether or not a temporal table will provide you the functionality you need to track how a record changes over time.

References

https://msdn.microsoft.com/en-nz/library/dn935015.aspx

https://msdn.microsoft.com/en-us/library/mt631669.aspx

https://msdn.microsoft.com/en-US/library/mt604468.aspx

https://msdn.microsoft.com/en-us/library/mt604469.aspx