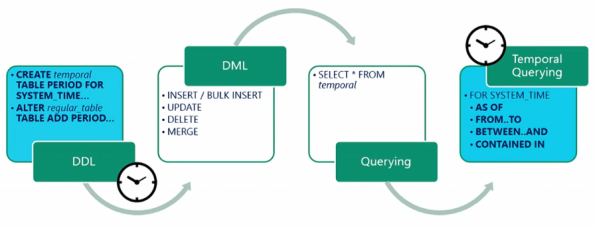
**Temporal data support in SQL Server 2016**

## Part 1

This coming release of new SQL Server 2016 will also support for **Temporal Data** as a new feature called **Temporal Tables** or **System-Versioned Tables**.

A **Temporal Data** is tightly related to a Time period and used to handle the time-varying data. It provides the ability to look at data trends, types of data changes, and the overall data evolution within your database.  
[](https://sqlwithmanoj.files.wordpress.com/2015/06/sql-server-2016-temporal-01.png)

A **Temporal Table** is a table for which a PERIOD definition exists and which contains system columns with a datatype of **datetime2** into which the period of validity is recorded by the system. The Temporal table has an associated **History Table** into which the system records all prior versions of each record with their period of validity.

With a Temporal Table, the value of each record at any **point in time** can be determined, rather than just the current value of each record.

The Temporal Data is closely related to **Slowly Changing Dimensions** (SCD) **Type-6** method, but the whole mechanism is internally and automatically managed by the SQL Server DB Engine.

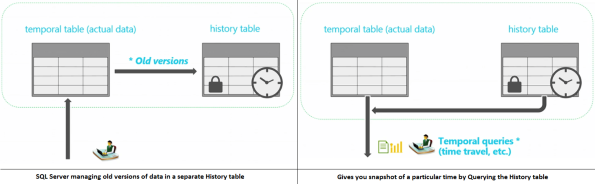
**–> With Temporal Data support you can:**

1. Time Travel back & forth and understand business trends over time.

2. Track and Audit all data changes.

3. Maintain a Slowly Changing Dimension (SCD) for decision support applications

4. Recover from accidental data changes by repairing record-level corruptions.

[](https://sqlwithmanoj.files.wordpress.com/2015/06/sql-server-2016-temporal-12.png)

**–>** To create a Temporal Table following clauses are used with CREATE TABLE statement:

1. Two additional Start & End date Audit columns of **datetime2** datatype for capturing the validity of records. You can use any meaningful column name here, we will use **ValidFrom** and **ValidTo** column names in our example below.

2. Both the column names have to be specified in and as **PERIOD FOR SYSTEM\_TIME (ValidFrom, ValidTo)** clause with the column list.

3. Specify **WITH (SYSTEM\_VERSIONING = ON)** option at the end of the CREATE TABLE statement with optional **(HISTORY\_TABLE = {History\_Table\_Name})** option.

The above CREATE TABLE statement will create 2 tables:

1. one the parent Temporal or System-Versioned Table

2. and second the History Table

**–>** Now, as I mentioned in Step #3 above, you have an option to specify the History Table name or not. So let’s check both the options here:

**–> Option #1:** Create Temporal Table [dbo].[Department] with **automatically named** History table:

USE [TestManDB]

GO

CREATE TABLE dbo.Department

(

    DepartmentID        int NOT NULL IDENTITY(1,1) PRIMARY KEY CLUSTERED,

    DepartmentName      varchar(50) NOT NULL,

    ManagerID           int NULL,

    ValidFrom           datetime2 GENERATED ALWAYS AS ROW START NOT NULL,

    ValidTo             datetime2 GENERATED ALWAYS AS ROW END   NOT NULL,

    PERIOD FOR SYSTEM\_TIME (

        ValidFrom,

        ValidTo

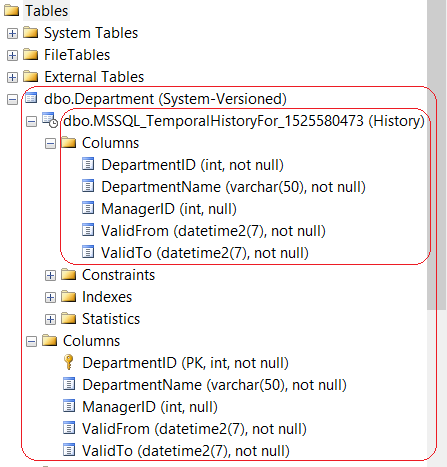
    )

)

WITH ( SYSTEM\_VERSIONING = ON ); -- No History table name given here

GO

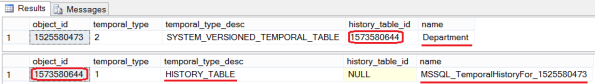
– Check the History Table nested under the Temporal Table dbo.Department . The auto-naming convention goes like this **MSSQL\_TemporalHistoryFor\_{parent\_temporal\_table\_object\_id}**:

[](https://sqlwithmanoj.files.wordpress.com/2015/06/sql-server-2016-temporal-04.png)

– Let’s Check the metadata of both the tables:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | SELECT object\_id, temporal\_type, temporal\_type\_desc, history\_table\_id, name -- Department  FROM SYS.TABLES  WHERE object\_id = OBJECT\_ID('dbo.Department', 'U')    SELECT object\_id, temporal\_type, temporal\_type\_desc, history\_table\_id, name -- MSSQL\_TemporalHistoryFor\_1397580017  FROM SYS.TABLES  WHERE object\_id = (      SELECT history\_table\_id      FROM SYS.TABLES      WHERE object\_id = OBJECT\_ID('dbo.Department', 'U')  )  GO |

– Output:

[](https://sqlwithmanoj.files.wordpress.com/2015/06/sql-server-2016-temporal-051.png)

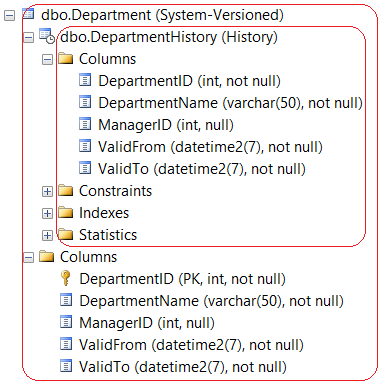
– To DROP both the Tables, first you need to switch OFF the System Versioning on the parent Temporal Table:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ALTER TABLE [dbo].[Department] SET ( SYSTEM\_VERSIONING = OFF )  GO  DROP TABLE [dbo].[Department]  GO  DROP TABLE [dbo].[MSSQL\_TemporalHistoryFor\_1525580473]  GO |

**–> Option #2:** Create Temporal Table [dbo].[Department] with a **named** History table [dbo].[DepartmentHistory]:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | CREATE TABLE dbo.Department  (      DepartmentID        int NOT NULL IDENTITY(1,1) PRIMARY KEY CLUSTERED,      DepartmentName      varchar(50) NOT NULL,      ManagerID           int NULL,        ValidFrom           datetime2 GENERATED ALWAYS AS ROW START NOT NULL,      ValidTo             datetime2 GENERATED ALWAYS AS ROW END   NOT NULL,        PERIOD FOR SYSTEM\_TIME (          ValidFrom,          ValidTo      )  )  WITH ( SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.DepartmentHistory) );  GO |

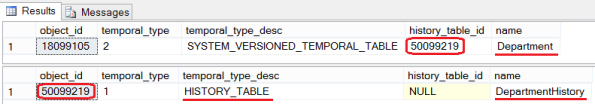
– Check the History Table dbo.DepartmentHistory nested under the parent Temporal Table dbo.Department:

[](https://sqlwithmanoj.files.wordpress.com/2015/06/sql-server-2016-temporal-061.png)

– Let’s Check the metadata of both the tables:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SELECT object\_id, temporal\_type, temporal\_type\_desc, history\_table\_id, name  FROM SYS.TABLES  WHERE object\_id = OBJECT\_ID('dbo.Department', 'U')    SELECT object\_id, temporal\_type, temporal\_type\_desc, history\_table\_id, name  FROM SYS.TABLES  WHERE object\_id = OBJECT\_ID('dbo.DepartmentHistory', 'U')  GO |

– Output:

[](https://sqlwithmanoj.files.wordpress.com/2015/06/sql-server-2016-temporal-071.png)

– Final Cleanup, As mentioned above to DROP both the Tables, first you need to switch OFF the System Versioning on the parent Temporal Table by ALTER statement:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ALTER TABLE [dbo].[Department] SET ( SYSTEM\_VERSIONING = OFF )  GO  DROP TABLE [dbo].[Department]  GO  DROP TABLE [dbo].[DepartmentHistory]  GO |

– Otherwise you will get following error message:

**Msg 13552, Level 16, State 1, Line 82  
Drop table operation failed on table ‘TestManDB.dbo.Department’ because it is not supported operation on system-versioned temporal tables.**

**–> Limitation of Temporal Tables:**

1. Temporal querying over **Linked Server** is not supported.

2. History table cannot have **constraints** (PK, FK, Table or Column constraints).

3. **INSERT** and **UPDATE** statements cannot reference the SYSTEM\_TIME period columns.

4. **TRUNCATE TABLE** is not supported while SYSTEM\_VERSIONING is ON

5. Direct **modification** of the data in a history table is not permitted.

6. **INSTEAD OF triggers** are not permitted on either the tables.

7. Usage of **Replication** technologies is limited.

## Part 2

**–>** Here in this post we will see how we can **enable System-Versioning** in an **existing table** containing data. I will also **tweak the data** to show you the demo on how you can point to a time back in **history** and get details relative to that time. This will be like **Time Travelling** to past and seeing record details as if its current data, without much change to the current table’s data-structure.

**1. Let’s create a sample table [dbo].[Employee] and populated it by JOINing some tables on [AdventureWorks] Database:**

USE [TestManDB]

GO

;WITH CTE AS (

    SELECT

        E.BusinessEntityID, P.FirstName, P.LastName, D.Name AS DepartmentName,

        ROW\_NUMBER() OVER(PARTITION BY E.BusinessEntityID ORDER BY D.ModifiedDate DESC) as RN

    FROM [AdventureWorks2014].[HumanResources].[Employee] E

    JOIN [AdventureWorks2014].[Person].[Person] P

    ON P.BusinessEntityID = E.BusinessEntityID

    JOIN [AdventureWorks2014].[HumanResources].[EmployeeDepartmentHistory] DH

    ON DH.BusinessEntityID = E.BusinessEntityID

    JOIN [AdventureWorks2014].[HumanResources].[Department] D

    ON D.DepartmentID = DH.DepartmentID

)

SELECT BusinessEntityID, FirstName, LastName, DepartmentName

    INTO dbo.Employee

FROM CTE

WHERE RN = 1

GO

**2. The System Versioned Temporal table [dbo].[Employee] must have Primary Key defined:** otherwise you will get the same error message (Msg 13553).

|  |  |
| --- | --- |
| 1  2  3 | ALTER TABLE dbo.Employee      ADD CONSTRAINT PK\_BusinessEntityID PRIMARY KEY (BusinessEntityID)  GO |

**3. Now to make [dbo].[Employee] table System Versioned we will add:**

– **Two Audit columns** of datetime2 datatype to store Start & End datetime.

– Use **PERIOD FOR SYSTEM\_TIME** clause to associate these two columns as System Time.

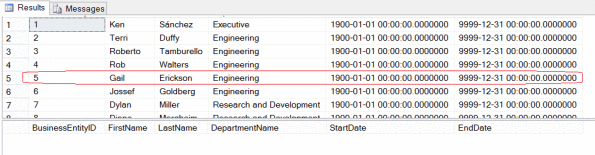
|  |  |
| --- | --- |
|  | ALTER TABLE dbo.Employee ADD      StartDate datetime2 GENERATED ALWAYS AS ROW START NOT NULL          DEFAULT CAST('1900-01-01 00:00:00.0000000' AS DATETIME2),      EndDate   datetime2 GENERATED ALWAYS AS ROW END   NOT NULL          DEFAULT CAST('9999-12-31 23:59:59.9999999' AS DATETIME2),  PERIOD FOR SYSTEM\_TIME (      StartDate,      EndDate  )  GO |

**4. After all pre-requisites let’s enable the System-Versioning on [dbo].[Employee] table:**

|  |  |
| --- | --- |
| 1  2  3 | ALTER TABLE dbo.Employee      SET (SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.EmployeeHistory))  GO |

**–>** So, as soon you enable the System-Versioning the SQL Engine creates an another History table with similar schema and nests it under the main Temporal table, let’s check both the tables columns and contents:

|  |  |
| --- | --- |
| 1  2  3 | SELECT TOP 10 \* FROM dbo.Employee  SELECT TOP 10 \* FROM dbo.EmployeeHistory  GO |

[](https://sqlwithmanoj.files.wordpress.com/2015/06/sql-server-2016-temporal-08.png)

You can see above the History table is empty as there are no updates on the Parent table. I’ve rounded the 5th row because I will update this row in next step for the demo.

**5. Let’s make some updates on the parent Temporal Table (5th row):** SQL Engine will automatically populate the History table.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | UPDATE dbo.Employee  SET FirstName = 'Gabriel'  WHERE BusinessEntityID = 5  GO    UPDATE dbo.Employee  SET DepartmentName = 'Research and Development'  WHERE BusinessEntityID = 5  GO    UPDATE dbo.Employee  SET DepartmentName = 'Executive'  WHERE BusinessEntityID = 5  GO    -- Let's check the records again:  SELECT \* FROM dbo.Employee WHERE BusinessEntityID = 5  SELECT \* FROM dbo.EmployeeHistory WHERE BusinessEntityID = 5  GO |

You will see that after doing 3 UPDATEs on the parent Temporal Table the History table [dbo].[EmployeeHistory] is populated with 3 rows that contains the older versions on data in [dbo].[Employee] table across all columns.

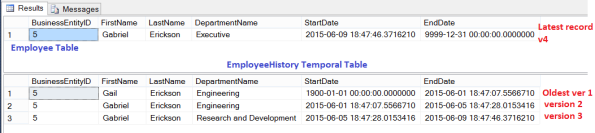
**6. Ok, now I’ll do some tweaks on the System Time column values of [dbo].[Employee] table.**

– First of all I will switch OFF the System-Versioning on dbo.Employee table.

– Now I will update the date of System Time columns, set it to back in history (5-10 days back for an effective demo).

– Enable back the System-Versioning

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | ALTER TABLE [dbo].[Employee] SET ( SYSTEM\_VERSIONING = OFF )  GO    update dbo.EmployeeHistory  set EndDate = '2015-06-01 18:47:07.5566710'  where BusinessEntityID = 5 AND EndDate = '2015-06-09 18:47:07.5566710'    update dbo.EmployeeHistory  set StartDate = '2015-06-01 18:47:07.5566710',      EndDate = '2015-06-05 18:47:28.0153416'  where BusinessEntityID = 5 AND StartDate = '2015-06-09 18:47:07.5566710'    update dbo.EmployeeHistory  set StartDate = '2015-06-05 18:47:28.0153416'  where BusinessEntityID = 5 AND StartDate = '2015-06-09 18:47:28.0153416'  GO    ALTER TABLE [dbo].[Employee]      SET (SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.EmployeeHistory))  GO    -- Check the data after datetime changes:  SELECT \* FROM dbo.Employee WHERE BusinessEntityID = 5  SELECT \* FROM dbo.EmployeeHistory WHERE BusinessEntityID = 5  GO |

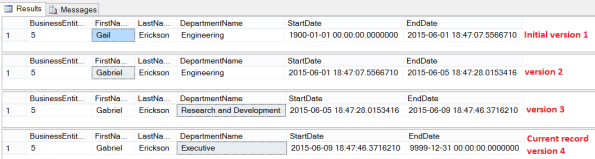
[](https://sqlwithmanoj.files.wordpress.com/2015/06/sql-server-2016-temporal-09.png)

**–> Ok, now here comes the most interesting part of the topic i.e. “Time Travel”:**

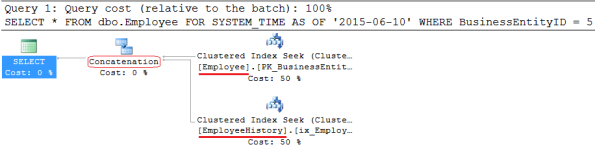
SQL Server engine provides you **FOR SYSTEM\_TIME AS OF {datetime\_value}** option with your SELECT query to get details pointing to any time in the history, let’s check this here:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | SELECT \*  FROM dbo.Employee  FOR SYSTEM\_TIME AS OF '2015-01-01'  WHERE BusinessEntityID = 5    SELECT \*  FROM dbo.Employee  FOR SYSTEM\_TIME AS OF '2015-06-03'  WHERE BusinessEntityID = 5    SELECT \*  FROM dbo.Employee  FOR SYSTEM\_TIME AS OF '2015-06-07'  WHERE BusinessEntityID = 5    SELECT \*  FROM dbo.Employee  FOR SYSTEM\_TIME AS OF '2015-06-10'  WHERE BusinessEntityID = 5  GO |

The above four Queries will show you results from the History Table by pulling out the records for the particular date you mentioned by checking the date ranges in the Audit columns:

[](https://sqlwithmanoj.files.wordpress.com/2015/06/sql-server-2016-temporal-10.png)

**–> Let’s check the Execution Plan of the 4th SELECT statement:**

[](https://sqlwithmanoj.files.wordpress.com/2015/06/sql-server-2016-temporal-111.png)

So, you can see that you are just querying the dbo.Employee Temporal table, but SQL Engine is internally also querying the dbo.EmployeeHistory table and concatenating (UNION) the rows from both the operators.

**–> Final Cleanup:** before Dropping the Temporal & History tables, you need to **switch OFF the System Versioning** feature in the parent table.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ALTER TABLE [dbo].[Employee] SET ( SYSTEM\_VERSIONING = OFF )  GO  DROP TABLE [dbo].[Employee]  GO  DROP TABLE [dbo].[EmployeeHistory]  GO |