





Creating a System-Versioned Temporal Table

SQL Server 2016 and later

Updated: May 24, 2016

Applies To: SQL Server 2016

THIS TOPIC APPLIES TO:  SQL Server (starting with 2016)  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

There are three ways to create a system-versioned temporal table with regards to how the history table is specified:

- Temporal table with an anonymous history table: you specify the schema of the current table and let the system create a corresponding history table with auto-generated name.
- Temporal table with a default history table: you specify the history table schema name and table name and let the system create a history table in that schema.
- Temporal table with a user-defined history table created beforehand: you create a history table that fits best your needs and then reference that table during temporal table creation.

Creating a temporal table with an anonymous history table

Creating a temporal table with an "anonymous" history table is a convenient option for quick object creation, especially in prototypes and test environments. It is also the simplest way to create a temporal table since it doesn't require any parameter in **SYSTEM_VERSIONING** clause. In the example below, a new table is created with system-versioning enabled without defining the name of the history table.

```
CREATE TABLE Department
(
    DeptID int NOT NULL PRIMARY KEY CLUSTERED
    , DeptName varchar(50) NOT NULL
    , ManagerID INT NULL
    , ParentDeptID int NULL
    , 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)
;
```

Important remarks

- A system-versioned temporal table must have a primary key defined and have exactly one **PERIOD FOR SYSTEM_TIME** defined with two datetime2 columns, declared as **GENERATED ALWAYS AS ROW START / END**
- The **PERIOD** columns are always assumed to be non-nullable, even if nullability is not specified. If the **PERIOD** columns are explicitly defined as nullable, the **CREATE TABLE** statement will fail.
- The history table must always be schema-aligned with the current or temporal table, in terms of number of columns, column names, ordering and data types.
- An anonymous history table is automatically created in the same schema as current or temporal table.
- The anonymous history table name has the following format:
MSSQL_TemporalHistoryFor_<current_temporal_table_object_id>_[suffix]. Suffix is optional and it will be added only if the first part of the table name is not unique.
- The history table is created as a rowstore table. PAGE compression is applied if possible, otherwise the history table will be uncompressed. For example, some table configurations, such as SPARSE columns, do not allow compression.
- A default clustered index is created for the history table with an auto-generated name in format *IX_<history_table_name>*. The clustered index contains the **PERIOD** columns (end, start).
- To create the current table as a memory-optimized table, see [System-Versioned Temporal Tables with Memory-Optimized Tables](#).

Creating a temporal table with a default history table

Creating a temporal table with a default history table is a convenient option when you want to control naming and still rely on the system to create the history table with the default configuration. In the example below, a new table is created with system-versioning enabled with the name of the history table explicitly defined.

```
CREATE TABLE Department
(
    DeptID int NOT NULL PRIMARY KEY CLUSTERED
    , DeptName varchar(50) NOT NULL
    , ManagerID INT NULL
    , ParentDeptID int NULL
    , 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.DepartmentHistory)
)
;
```

Important remarks

The history table is created using the same rules as apply to creating an "anonymous" history table, with the following rules that apply specifically to the named history table.

- The schema name is mandatory for the **HISTORY_TABLE** parameter.
- If the specified schema does not exist, the **CREATE TABLE** statement will fail.
- If the table specified by the **HISTORY_TABLE** parameter already exists, it will be validated against the newly created temporal table in terms of [schema consistency and temporal data consistency](#). If you specify an invalid history table, the **CREATE TABLE** statement will fail.

Creating a temporal table with a user-defined history table

Creating a temporal table with user-defined history table is a convenient option when the user wants to specify a history table with specific storage options and additional indexes. In the example below, a user-defined history table is created with a schema that is aligned with the temporal table that will be created. To this user-defined history table, a clustered columnstore index and additional non clustered rowstore (B-tree) index is created for point lookups. After this user-defined history table is created, the system-versioned temporal table is created specifying the user-defined history table as the default history table.

```
CREATE TABLE DepartmentHistory
(
    DeptID int NOT NULL
    , DeptName varchar(50) NOT NULL
    , ManagerID INT NULL
    , ParentDeptID int NULL
    , SysStartTime datetime2 NOT NULL
    , SysEndTime datetime2 NOT NULL
);
GO
CREATE CLUSTERED COLUMNSTORE INDEX IX_DepartmentHistory
    ON DepartmentHistory;
CREATE NONCLUSTERED INDEX IX_DepartmentHistory_ID_PERIOD_COLUMNS
    ON DepartmentHistory (SysEndTime, SysStartTime, DeptID);
GO
CREATE TABLE Department
(
    DeptID int NOT NULL PRIMARY KEY CLUSTERED
    , DeptName varchar(50) NOT NULL
    , ManagerID INT NULL
    , ParentDeptID int NULL
    , 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.DepartmentHistory))
;
```

Important remarks

- If you plan to run analytic queries on the historical data that employs aggregates or windowing functions, creating a clustered columnstore as a primary index is highly recommended for compression and query performance.
- If the primary use case is data audit (i.e. searching for historical changes for a single row from the current table), then a good choice is to create rowstore history table with a clustered index
- The history table cannot have a primary key, foreign keys, unique indexes, table constraints or triggers. It cannot be configured for change data capture, change tracking, transactional or merge replication.

Alter Non-Temporal Table to be System-Versioned Temporal Table

When you need to enable system-versioning using an existing table, such as when you wish to migrate a custom temporal solution to built-in support.

For example, you may have a set of tables where versioning is implemented with triggers. Using temporal system-versioning is less complex and provides additional benefits including:

- immutable history

- new syntax for time-travelling queries
- better DML performance
- minimal maintenance costs

When converting an existing table, consider using the **HIDDEN** clause to hide the new **PERIOD** columns to avoid impacting existing applications that are not designed to handle new columns.

Adding versioning to non-temporal tables

If you want to start tracking changes for a non-temporal table that contains the data, you need to add the **PERIOD** definition and optionally provide a name for the empty history table that SQL Server will create for you:

```
CREATE SCHEMA History;
GO
ALTER TABLE InsurancePolicy
    ADD
        SysStartTime datetime2(0) GENERATED ALWAYS AS ROW START HIDDEN
            CONSTRAINT DF_SysStart DEFAULT SYSUTCDATETIME()
        , SysEndTime datetime2(0) GENERATED ALWAYS AS ROW END HIDDEN
            CONSTRAINT DF_SysEnd DEFAULT CONVERT(datetime2 (0), '9999-12-31 23:59:59'),
        PERIOD FOR SYSTEM_TIME (SysStartTime, SysEndTime);
GO
ALTER TABLE InsurancePolicy
    SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE = History.InsurancePolicy))
;
```

Important remarks

- Adding non-nullable columns with defaults to an existing table with data is a size of data operation on all editions other than SQL Server Enterprise Edition (on which it is a metadata operation). With a large existing history table with data on SQL Server Standard Edition, adding a non-null column can be an expensive operation.
- Constraints for period start and period end columns must be carefully chosen:
 - Default for start column specifies from which point in time you consider existing rows to be valid. It cannot be specified as a datetime point in the future.
 - End time must be specified as the maximum value for a given datetime2 precision
- Adding period will perform a data consistency check on the current table to make sure that the defaults for period columns are valid.
- When an existing history table is specified when enabling **SYSTEM_VERSIONING**, a data consistency check will be performed across both the current and the history table. It can be skipped if you specify **DATA_CONISTENCY_CHECK = OFF** as an additional parameter.

Migrate existing tables to built-in support

This example shows how to migrate an existing solution based on triggers to build-in temporal support. For this example, we

assume that the current custom solution splits the current and historical data in two separate user tables (**ProjectTaskCurrent** and **ProjectTaskHistory**). If your existing solution uses single table to store actual and historical rows, then you should split the data into two tables prior to the migration steps shown in this example:

```
/*Drop trigger on future temporal table*/
DROP TRIGGER ProjectCurrent_OnUpdateDelete;
/*Make sure that future period columns are non-nullable*/
ALTER TABLE ProjectTaskCurrent ALTER COLUMN [ValidFrom] datetime2 NOT NULL;
ALTER TABLE ProjectTaskCurrent ALTER COLUMN [ValidTo] datetime2 NOT NULL;
ALTER TABLE ProjectTaskHistory ALTER COLUMN [ValidFrom] datetime2 NOT NULL;
ALTER TABLE ProjectTaskHistory ALTER COLUMN [ValidTo] datetime2 NOT NULL;
ALTER TABLE ProjectTaskCurrent
    ADD PERIOD FOR SYSTEM_TIME ([ValidFrom], [ValidTo])
ALTER TABLE ProjectTaskCurrent
    SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.ProjectTaskHistory,
    DATA_CONSISTENCY_CHECK = ON))
;
```

Important remarks

- Referencing existing columns in **PERIOD** definition implicitly changes generated_always_type to **AS_ROW_START** and **AS_ROW_END** for those columns.
- Adding **PERIOD** will perform a data consistency check on current table to make sure that the existing values for period columns are valid
- It is highly recommended to set **SYSTEM_VERSIONING** with **DATA_CONSISTENCY_CHECK = ON** to enforce data consistency checks on existing data.

Did this Article Help You? We're Listening

What information are you looking for, and did you find it? We're listening to your feedback to improve the content. Please submit your comments to sqlfeedback@microsoft.com

See Also

[Temporal Tables](#)

[Getting Started with System-Versioned Temporal Tables](#)

[Manage Retention of Historical Data in System-Versioned Temporal Tables](#)

[System-Versioned Temporal Tables with Memory-Optimized Tables](#)

[CREATE TABLE \(Transact-SQL\)](#)

[Modifying Data in a System-Versioned Temporal Table](#)

[Querying Data in a System-Versioned Temporal Table](#)

[Changing the Schema of a System-Versioned Temporal Table](#)

[Stopping System-Versioning on a System-Versioned Temporal Table](#)

Community Additions

HIDDEN option in Temporal Table

Hello there,

I'm trying to figure out, what for is HIDDEN option. When I create temporal table without this option, INSERT statement works with/without specifying system version columns. Also option DROP and CREATE table from right click menu on table, doesn't script table with option HIDDEN even if it was created before with HIDDEN option. Could you please clarify more?

Thanks,

Andrej Zafka



Andrej Zafka

10/16/2016

© 2017 Microsoft