<https://learn.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables?view=sql-server-ver16>

# Temporal tables

* Article
* 10/16/2023
* 13 contributors

Feedback

## In this article

1. [What is a system-versioned temporal table?](https://learn.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables?view=sql-server-ver16#what-is-a-system-versioned-temporal-table)
2. [Why temporal?](https://learn.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables?view=sql-server-ver16#why-temporal)
3. [How does temporal work?](https://learn.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables?view=sql-server-ver16#how-does-temporal-work)
4. [How do I query temporal data?](https://learn.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables?view=sql-server-ver16#how-do-i-query-temporal-data)

Show 3 more

**Applies to:**  SQL Server 2016 (13.x) and later  [Azure SQL Database](https://learn.microsoft.com/en-us/sql/sql-server/sql-docs-navigation-guide?view=sql-server-ver16#applies-to)  [Azure SQL Managed Instance](https://learn.microsoft.com/en-us/sql/sql-server/sql-docs-navigation-guide?view=sql-server-ver16#applies-to)

Temporal tables (also known as system-versioned temporal tables) are a database feature that brings built-in support for providing information about data stored in the table at any point in time, rather than only the data that is correct at the current moment in time.

[Get started with system-versioned temporal tables](https://learn.microsoft.com/en-us/sql/relational-databases/tables/getting-started-with-system-versioned-temporal-tables?view=sql-server-ver16), and review [Temporal table usage scenarios](https://learn.microsoft.com/en-us/sql/relational-databases/tables/temporal-table-usage-scenarios?view=sql-server-ver16).

## What is a system-versioned temporal table?W

A system-versioned temporal table is a type of user table designed to keep a full history of data changes, allowing easy point-in-time analysis. This type of temporal table is referred to as a system-versioned temporal table because the period of validity for each row is managed by the system (that is, the database engine).

Every temporal table has two explicitly defined columns, each with a **datetime2** data type. These columns are referred to as period columns. These period columns are used exclusively by the system to record the period of validity for each row, whenever a row is modified. The main table that stores current data is referred to as the current table, or simply as the temporal table.

In addition to these period columns, a temporal table also contains a reference to another table with a mirrored schema, called the history table. The system uses the history table to automatically store the previous version of the row each time a row in the temporal table gets updated or deleted. During temporal table creation users can specify an existing history table (which must be schema compliant) or let the system create a default history table.

## Why temporal?

Real data sources are dynamic and more often than not business decisions rely on insights that analysts can get from data evolution. Use cases for temporal tables include:

* Auditing all data changes and performing data forensics when necessary
* Reconstructing state of the data as of any time in the past
* Calculating trends over time
* Maintaining a slowly changing dimension for decision support applications
* Recovering from accidental data changes and application errors

## How does temporal work?

System-versioning for a table is implemented as a pair of tables, a current table and a history table. Within each of these tables, two additional **datetime2** columns are used to define the period of validity for each row:

* Period start column: The system records the start time for the row in this column, typically denoted as the ValidFrom column.
* Period end column: The system records the end time for the row in this column, typically denoted as the ValidTo column.

The current table contains the current value for each row. The history table contains each previous value (the old version) for each row, if any, and the start time and end time for the period for which it was valid.

The following script illustrates a scenario with employee information:

SQLCopy

CREATE TABLE dbo.Employee (

[EmployeeID] INT NOT NULL PRIMARY KEY CLUSTERED,

[Name] NVARCHAR(100) NOT NULL,

[Position] VARCHAR(100) NOT NULL,

[Department] VARCHAR(100) NOT NULL,

[Address] NVARCHAR(1024) NOT NULL,

[AnnualSalary] DECIMAL(10, 2) NOT NULL,

[ValidFrom] DATETIME2 GENERATED ALWAYS AS ROW START,

[ValidTo] DATETIME2 GENERATED ALWAYS AS ROW END,

PERIOD FOR SYSTEM\_TIME(ValidFrom, ValidTo)

)

WITH (SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.EmployeeHistory));

For more information, see [Create a system-versioned temporal table](https://learn.microsoft.com/en-us/sql/relational-databases/tables/creating-a-system-versioned-temporal-table?view=sql-server-ver16).

* **Inserts:** The system sets the value for the ValidFrom column to the begin time of the current transaction (in the UTC time zone) based on the system clock and assigns the value for the ValidTo column to the maximum value of 9999-12-31. This marks the row as open.
* **Updates:** The system stores the previous value of the row in the history table and sets the value for the ValidTo column to the begin time of the current transaction (in the UTC time zone) based on the system clock. This marks the row as closed, with a period recorded for which the row was valid. In the current table, the row is updated with its new value and the system sets the value for the ValidFrom column to the begin time for the transaction (in the UTC time zone) based on the system clock. The value for the updated row in the current table for the ValidTo column remains the maximum value of 9999-12-31.
* **Deletes:** The system stores the previous value of the row in the history table and sets the value for the ValidTo column to the begin time of the current transaction (in the UTC time zone) based on the system clock. This marks the row as closed, with a period recorded for which the previous row was valid. In the current table, the row is removed. Queries of the current table don't return this row. Only queries that deal with history data return data for which a row is closed.
* **Merge:** The operation behaves exactly as if up to three statements (an INSERT, an UPDATE, and/or a DELETE) executed, depending on what is specified as actions in the MERGE statement.

**Important**

The times recorded in the system **datetime2** columns are based on the begin time of the transaction itself. For example, all rows inserted within a single transaction have the same UTC time recorded in the column corresponding to the start of the SYSTEM\_TIME period.

**Note**

When you run any data modification queries on a temporal table, the Database Engine adds a row to the history table even if no column values change.

## How do I query temporal data?

The SELECT ... FROM <table> statement has a new clause FOR SYSTEM\_TIME, with five temporal-specific subclauses to query data across the current and history tables. This new SELECT statement syntax is supported directly on a single table, propagated through multiple joins, and through views on top of multiple temporal tables.

When you query using the FOR SYSTEM\_TIME clause using one of the five subclauses, historical data from the temporal table are included, as shown in the following image.

The following query searches for row versions for an employee with the filter condition WHERE EmployeeID = 1000 that were active at least for a portion of period between January 1, 2021 and January 1, 2022 (including the upper boundary):

SQLCopy

SELECT \* FROM Employee

FOR SYSTEM\_TIME

BETWEEN '2021-01-01 00:00:00.0000000' AND '2022-01-01 00:00:00.0000000'

WHERE EmployeeID = 1000 ORDER BY ValidFrom;

**Note**

FOR SYSTEM\_TIME filters out rows that have a period of validity with zero duration (ValidFrom = ValidTo).

Those rows will be generated if you perform multiple updates on the same primary key within the same transaction. In that case, temporal querying returns only row versions before the transactions, and current rows after the transactions.

If you need to include those rows in the analysis, query the history table directly.

In the following table, ValidFrom in the Qualifying Rows column represents the value in the ValidFrom column in the table being queried and ValidTo represents the value in the ValidTo column in the table being queried. For the full syntax and for examples, see [FROM (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/queries/from-transact-sql?view=sql-server-ver16) and [Query data in a system-versioned temporal table](https://learn.microsoft.com/en-us/sql/relational-databases/tables/querying-data-in-a-system-versioned-temporal-table?view=sql-server-ver16).

Expand table

| **Expression** | **Qualifying Rows** | **Note** |
| --- | --- | --- |
| AS OF date\_time | ValidFrom <= date\_time AND ValidTo > date\_time | Returns a table with rows containing the values that were current at the specified point in time in the past. Internally, a union is performed between the temporal table and its history table and the results are filtered to return the values in the row that was valid at the point in time specified by the date\_time parameter. The value for a row is deemed valid if the system\_start\_time\_column\_name value is less than or equal to the date\_time parameter value and the system\_end\_time\_column\_name value is greater than the date\_time parameter value. |
| FROM start\_date\_time TO end\_date\_time | ValidFrom < end\_date\_time AND ValidTo > start\_date\_time | Returns a table with the values for all row versions that were active within the specified time range, regardless of whether they started being active before the start\_date\_time parameter value for the FROM argument or ceased being active after the end\_date\_time parameter value for the TO argument. Internally, a union is performed between the temporal table and its history table. The results are filtered to return the values for all row versions that were active at any time during the time range specified. Rows that stopped being active exactly on the lower boundary defined by the FROM endpoint aren't included, and records that became active exactly on the upper boundary defined by the TO endpoint are also not included. |
| BETWEEN start\_date\_time AND end\_date\_time | ValidFrom <= end\_date\_time AND ValidTo > start\_date\_time | Same as previous in the FOR SYSTEM\_TIME FROM start\_date\_time TO end\_date\_time description, except the table of rows returned includes rows that became active on the upper boundary defined by the end\_date\_time endpoint. |
| CONTAINED IN (start\_date\_time, end\_date\_time) | ValidFrom >= start\_date\_time AND ValidTo <= end\_date\_time | Returns a table with the values for all row versions that were opened and closed within the specified time range defined by the two period values for the CONTAINED IN argument. Rows that became active exactly on the lower boundary or ceased being active exactly on the upper boundary are included. |
| ALL | All rows | Returns the union of rows that belong to the current and the history table. |

## Hide the period columns

You can choose to hide the period columns, such that queries that don't explicitly reference them don't return these columns (for example, when running SELECT \* FROM <table>).

To return a hidden column, you must explicitly refer to the hidden column in the query. Similarly INSERT and BULK INSERT statements continue as if these new period columns weren't present (and the column values are auto-populated).

For details on using the HIDDEN clause, see [CREATE TABLE](https://learn.microsoft.com/en-us/sql/t-sql/statements/create-table-transact-sql?view=sql-server-ver16) and [ALTER TABLE](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-table-transact-sql?view=sql-server-ver16).

## Samples

### ASP.NET

See this [ASP.NET Core web application](https://github.com/microsoft/sql-server-samples/blob/master/samples/features/temporal/) to learn how to build a temporal application using temporal tables.

### Download Adventure Works sample database

You can download the [AdventureWorks database for SQL Server](https://github.com/microsoft/sql-server-samples/releases/tag/adventureworks), which includes temporal table features.

## Related content

* [Temporal table considerations and limitations](https://learn.microsoft.com/en-us/sql/relational-databases/tables/temporal-table-considerations-and-limitations?view=sql-server-ver16)
* [Manage retention of historical data in system-versioned temporal tables](https://learn.microsoft.com/en-us/sql/relational-databases/tables/manage-retention-of-historical-data-in-system-versioned-temporal-tables?view=sql-server-ver16)
* [Partitioning with temporal tables](https://learn.microsoft.com/en-us/sql/relational-databases/tables/partitioning-with-temporal-tables?view=sql-server-ver16)
* [Temporal table system consistency checks](https://learn.microsoft.com/en-us/sql/relational-databases/tables/temporal-table-system-consistency-checks?view=sql-server-ver16)
* [Temporal Table Security](https://learn.microsoft.com/en-us/sql/relational-databases/tables/temporal-table-security?view=sql-server-ver16)
* [Temporal Table Metadata Views and Functions](https://learn.microsoft.com/en-us/sql/relational-databases/tables/temporal-table-metadata-views-and-functions?view=sql-server-ver16)
* [Working with Memory-Optimized System-Versioned Temporal Tables](https://learn.microsoft.com/en-us/sql/relational-databases/tables/working-with-memory-optimized-system-versioned-temporal-tables?view=sql-server-ver16)
* [Create a system-versioned temporal table](https://learn.microsoft.com/en-us/sql/relational-databases/tables/creating-a-system-versioned-temporal-table?view=sql-server-ver16)
* [Modifying data in a system-versioned temporal table](https://learn.microsoft.com/en-us/sql/relational-databases/tables/modifying-data-in-a-system-versioned-temporal-table?view=sql-server-ver16)
* [Query data in a system-versioned temporal table](https://learn.microsoft.com/en-us/sql/relational-databases/tables/querying-data-in-a-system-versioned-temporal-table?view=sql-server-ver16)
* [Getting Started with system-versioned temporal tables](https://learn.microsoft.com/en-us/sql/relational-databases/tables/getting-started-with-system-versioned-temporal-tables?view=sql-server-ver16)
* [System-Versioned Temporal Tables with Memory-Optimized Tables](https://learn.microsoft.com/en-us/sql/relational-databases/tables/system-versioned-temporal-tables-with-memory-optimized-tables?view=sql-server-ver16)
* [Getting Started with Temporal Tables in Azure SQL Database](https://learn.microsoft.com/en-us/azure/azure-sql/temporal-tables)