Temporale Tabellen

Motivation

- Daten in Tabellen, deren Änderungsverlauf von Interesse ist
  - Für Statistik, Datenforensik, Wiederherstellung bei fehlerhaften Datenänderungen, ...
  - Lückenloses Speichern von Datenänderungen
- Vor SQL Server 2016:
  - Zwei Tabellen mit einer 1:n-Beziehung + Trigger
  - Unübersichtlich, Abfrage von Verlaufsdaten komplizierter
- Ab SQL Server 2016:
  - System-versioned tables / Temporale Tabellen
  - Abfragen über Schlüsselwörter
  - Übersichtlicher, einfacher

Erstellen einer Temporalen Tabelle

- Im Prinzip handelt es sich um zwei Tabellen (Haupttabelle und Verlaufstabelle)
- Zwei Spalten vom Datentyp DATETIME2 sind Pflicht
  - Notwendig um den Gültigkeitszeitraum abzugrenzen
- Ein definierter Primärschlüssel ist Pflicht
- Die Verlaufstabelle kann man:
  - Automatisch mit von der Haupttabelle kopiertem Schema erstellen lassen
  - Benutzerdefiniert erstellen und der Haupttabelle zuweisen
- Bei Änderung von Daten der Haupttabelle wird diese automatisch gefüllt

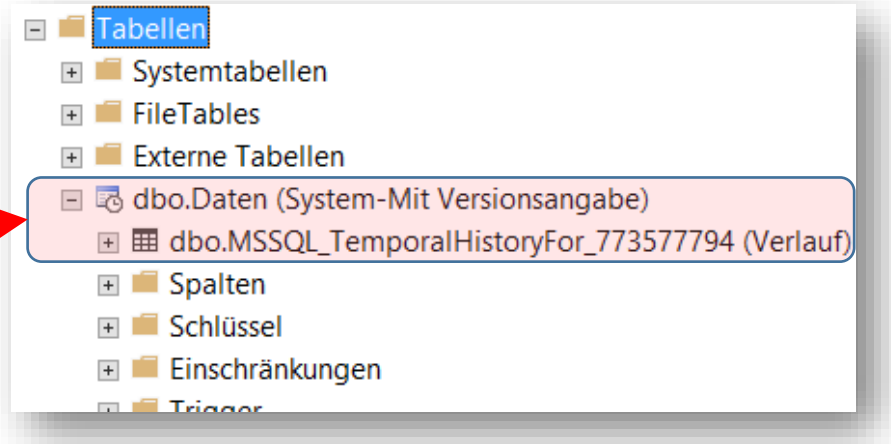
Beispiel I – automatisch erstellte Verlaufstabelle

```
CREATE TABLE Daten
(
    Daten_ID        INT
        CONSTRAINT PK_Daten_ID PRIMARY KEY IDENTITY,
    Daten           NVARCHAR(MAX),
    StartDatum          DATETIME2
        GENERATED ALWAYS AS ROW START   DEFAULT SYSUTCDATETIME(),
    EndDatum            DATETIME2
        GENERATED ALWAYS AS ROW END     DEFAULT CONVERT(DATETIME2, '9999-12-31 23:59:59'),
    PERIOD FOR SYSTEM_TIME(StartDatum, EndDatum)
)
WITH (SYSTEM_VERSIONING = ON);
```

Zwei Spalten vom Typ **DATETIME2**

SQL Server muss wissen, welche zwei Spalten für die Zeiträume definiert sind

Aktivieren der Versionierungsverwaltung



Haupttabelle und Verlaufstabelle

Eigenschaften

- Die Haupttabelle kann nicht gelöscht werden, solange **SYSTEM_VERSIONING** = **ON** ist
  - Zum Löschen ändern mit **ALTER TABLE** <Tabellenname> **SET** (**SYSTEM_VERSIONING** = **OFF**)
  - Beim Löschen bleibt die Verlaufstabelle erhalten

- Den Namen der Verlaufstabelle kann man auch selbst bestimmen

```
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.DatenHistorie));
```

- Es ist kein INSTEAD OF-Trigger erlaubt
  - Microsoft hat offensichtlich schon einen INSTEAD OF-Trigger für die Versionierung angelegt

Beispiel II – benutzerdefinierte Verlaufstabelle

```
CREATE TABLE DatenHistorie2
(
    Daten_ID INT NOT NULL,
    Daten NVARCHAR(MAX),
    Startpunkt DATETIME2 NOT NULL,
    Endpunkt DATETIME2 NOT NULL
);
```

Erstellen einer Verlaufstabelle

Erstellen der Haupttabelle und Zuweisen der HISTORY_TABLE

```
CREATE TABLE Daten2
(
    Daten_ID        INT
        CONSTRAINT PK_Daten_ID2 PRIMARY KEY IDENTITY,
    Daten           NVARCHAR(MAX),
    Startpunkt          DATETIME2
        GENERATED ALWAYS AS ROW START   DEFAULT SYSUTCDATETIME(),
    Endpunkt            DATETIME2
        GENERATED ALWAYS AS ROW END     DEFAULT CONVERT(DATETIME2, '9999-12-31 23:59:59'),
    PERIOD FOR SYSTEM_TIME(Startpunkt, Endpunkt)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.DatenHistorie2));
```

- Die Spaltennamen, Datentypen und NOT NULL-Eigenschaften müssen mit der erstellten Verlaufstabelle übereinstimmen, ansonsten Fehlermeldung

INSERT, UPDATE, DELETE

INSERT

- Nur die temporale Tabelle (Haupttabelle) bekommt neue Daten
- Startpunkt-Spalte bekommt den Anfangszeitpunkt der aktuellen Transaktion
- Endpunkt-Spalte bekommt den Maximalwert (Datensatz ist geöffnet)

UPDATE

- Die ursprüngliche Zeile kommt in die Verlaufstabelle mit aktuellem Zeitwert in der Endpunkt-Spalte
- In der Haupttabelle werden die Daten aktualisiert
- In der Haupttabelle aktueller Zeitwert in Startpunkt-Spalte, Maximalwert in Endpunkt-Spalte

DELETE

- Die Verlaufstabelle bekommt den zu löschenden Datensatz mit aktueller Zeit in Endpunkt-Spalte (Datensatz ist geschlossen)
- Der Datensatz wird in der Haupttabelle gelöscht

MERGE und TRUNCATE

MERGE

- Hinter MERGE verbergen sich INSERT, UPDATE und DELETE
- Die entsprechenden Veränderungen sind äquivalent

TRUNCATE

- **TRUNCATE TABLE** <Tabellenname> ist nicht möglich, wenn die Versionierung eingeschaltet ist

Abfragen temporaler Daten

- Fünf Unterklauseln für die Abfrage temporaler Daten:
  - **ALL**, **AS OF**, **BETWEEN** … **AND**, **FROM** … **TO**, **CONTAINED IN**
  - Alle benötigen das Schlüsselwort **FOR SYSTEM_TIME**

ALL

- Gibt die Vereinigungsmenge der Daten beider Tabellen zurück

```
SELECT * FROM Daten
FOR SYSTEM_TIME ALL
WHERE Daten_ID = 5;
```

	Daten_ID	Daten	StartDatum	EndDatum
1	5	blubb	2018-01-10 12:21:43.7699239	9999-12-31 23:59:59.9999999
2	5	bla	2018-01-10 11:34:27.9365878	2018-01-10 12:21:43.7699239

AS OF

- Gibt die zu einem bestimmten Zeitpunkt gültigen Daten aus

```
SELECT * FROM Daten
FOR SYSTEM_TIME AS OF '2018-01-10 11:35'
WHERE Daten_ID = 5;

SELECT * FROM Daten
FOR SYSTEM_TIME AS OF '2018-01-10 12:22'
WHERE Daten_ID = 5;
```

	Daten_ID	Daten	StartDatum	EndDatum
1	5	bla	2018-01-10 11:34:27.9365878	2018-01-10 12:21:43.7699239

	Daten_ID	Daten	StartDatum	EndDatum
1	5	blubb	2018-01-10 12:21:43.7699239	9999-12-31 23:59:59.9999999

FROM … TO

- Gibt alle in einem angegebenen Zeitraum aktuellen Daten zurück

```
SELECT * FROM Daten
FOR SYSTEM_TIME
FROM '2018-01-10 11:35' TO '2018-01-10 13:00'
WHERE Daten_ID = 5;
```

	Daten_ID	Daten	StartDatum	EndDatum
1	5	blubb	2018-01-10 12:21:43.7699239	9999-12-31 23:59:59.9999999
2	5	bla	2018-01-10 11:34:27.9365878	2018-01-10 12:21:43.7699239

BETWEEN … AND

- Wie **FROM** … **TO**, aber die Intervallgrenzpunkte werden mit berücksichtigt
  - Irgendwer stellt jetzt die Frage „Ist das bei FROM … TO nicht so?“ → Genau!!

CONTAINED IN

- Gibt Zeilenversionen zurück, welche innerhalb zweier Zeitpunkte geöffnet und geschlossen wurden

```
SELECT * FROM Daten
FOR SYSTEM_TIME
CONTAINED IN ('2018-01-10 11:30', '2018-01-10 13:00');
```

	Daten_ID	Daten	StartDatum	EndDatum
1	5	bla	2018-01-10 11:34:27.9365878	2018-01-10 12:21:43.7699239
2	2	blablablabla	2018-01-10 11:34:27.9365878	2018-01-10 12:59:15.1902465