|  |  |
| --- | --- |
|  |  |
|  | SQL Server 2016 CTP2 Temporal HOL |

Contents

[Overview and setup 3](#_Toc421781278)

[Create temporal tables 4](#_Toc421781279)

[Explore temporal data and metadata 8](#_Toc421781280)

[Create temporal queries 11](#_Toc421781281)

[Manage and maintain temporal indexes 15](#_Toc421781282)

[Terms of use 17](#_Toc421781283)

|  |
| --- |
| Overview and setup |

### Estimated time to complete lab is 40-45 minutes.

### Overview

Data is rarely static. Sometimes it can be very valuable to see how data has evolved over time, or to query data as of a particular point in time. While traditional databases store the data that is considered to be valid at the current time only, SQL Server 2016 CTP2 Temporal features provide correct information about stored facts at any point in time.

Each Temporal table (or ‘system-versioned’ temporal table) consists of two tables actually – one for the current data and the other one for the historical data. Additional querying constructs are provided to hide this complexity from users.

Common scenarios for using Temporal data include:

* Querying data as of a particular point in time to examine the state of the data at that time;
* Providing regulatory compliance and perform auditing;
* Implementing slowly changing dimensions; and
* Reverting a table to “last good known state” without downtime;

This hands on lab will familiarize you with the new Temporal data structures and queries and help you understand how to implement them. In particular, you will learn:

1. How to create temporal tables;
2. How to explore temporal data and metadata;
3. How to run temporal queries;
4. How to manage and maintain temporal indexes.

This lab has four parts. The first part demonstrates scenarios for creating temporal tables or adding temporal functionality to existing tables. The second part involves examining temporal data and metadata. The third part of the lab looks at several scenarios for querying temporal data. Finally, we will investigate how to manage and maintain temporal indexes. At the end of this lab, you will have worked through several of the most common scenarios involved with the new Temporal features in SQL Server 2016 CTP2.

**Setting up your environment**

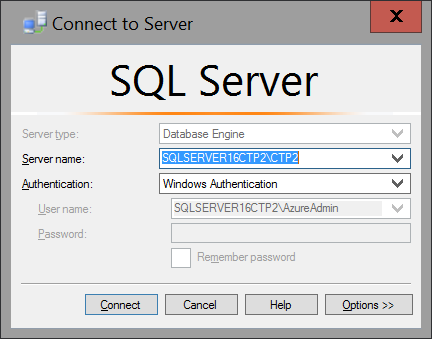
1. The script file that contains the scripts to be used in this lab is located in the **C:\SQL Server 2016 CPT2 HOLs\Temporal** folder.
2. Open **Notepad.exe**. You will use **Notepad** as a scratchpad tool for copy and paste of text elements.

|  |
| --- |
| Create temporal tables |

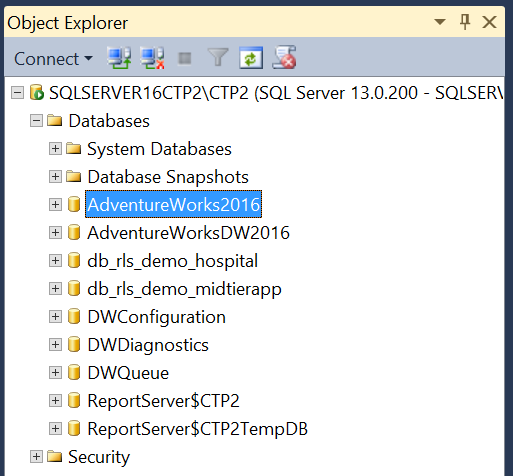
In this lab, we will explore creating temporal tables and to add temporal system versioning to existing tables. Let’s get started!

### Create a new temporal table using history table defaults

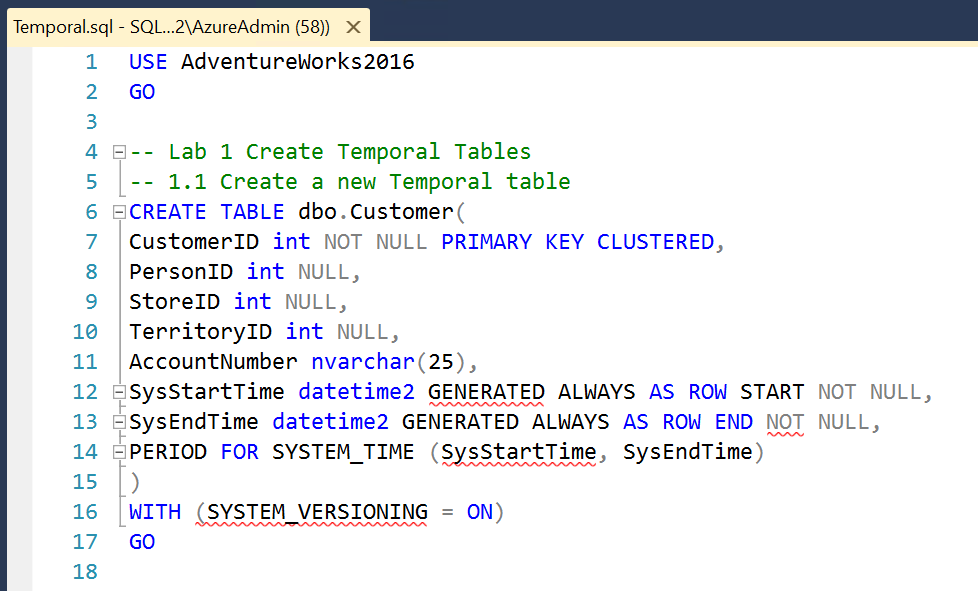
1. Open up **SQL Server Management Studio** and connect to the **SQLSERVER16CTP2\CTP2** Database Engine instance.



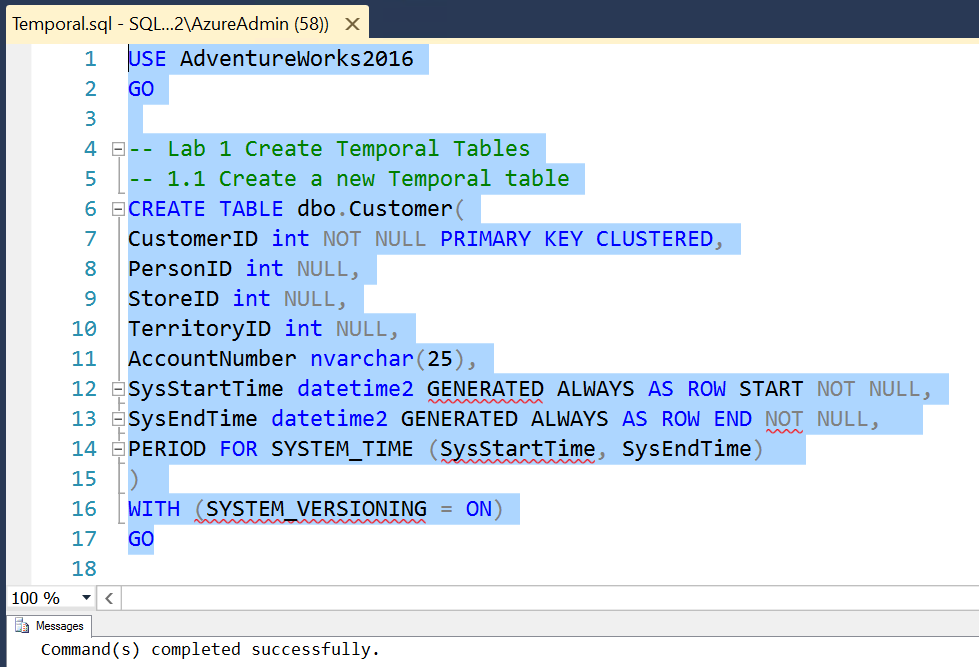
1. Expand the **Databases** folder in **Object Explorer** and click on **AdventureWorks2016** to select it.



1. Press **Ctrl+O** and open the file **C:\SQL Server 2016 CPT2 HOLs\Temporal\Temporal.sql**.

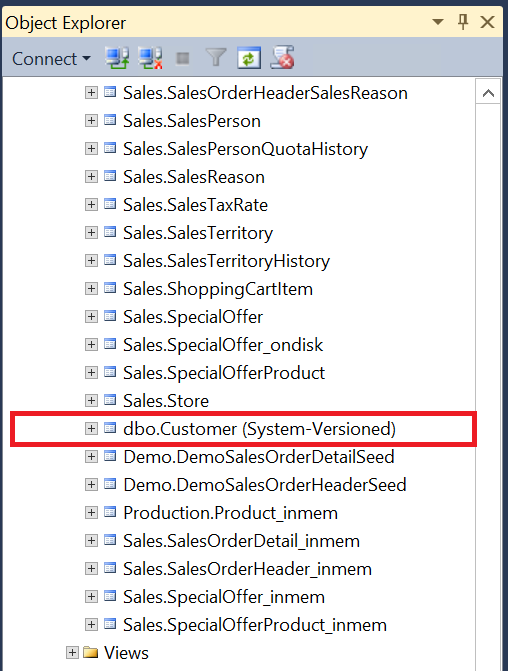


1. Select the **USE AdventureWorks2016** and **CREATE TABLE dbo.Customer** statements and press **F8** to create the new **b** table in the **dbo** schema within **AdventureWorks2016**.



Note that the table must use datetime2 as the data type for the temporal fields.

1. Expand out the list of **Tables** in the **Object Explorer** and scroll down the list until you see the new **dbo.Customer (System-Versioned)** table.



### Add temporal features to an existing table

We will manually add temporal to the Person.BusinessEntityContact table in AdventureWorks2016.

Note, the following SQL Statements are included in the **C:\SQL Server 2016 CPT2 HOLs\Temporal\Temporal.sql** script.

1. First, create a history table that mirrors the table you want to add temporal features to. Add **SysStartTime** and **SysEndTime** columns.

CREATE TABLE Person.BusinessEntityContactHistory(

BusinessEntityID int NOT NULL,

PersonID int NOT NULL,

ContactTypeID int NOT NULL,

rowguid uniqueidentifier ROWGUIDCOL NOT NULL,

ModifiedDate datetime NOT NULL,

--Add new columns

SysStartTime datetime2 NOT NULL,

SysEndTime datetime2 NOT NULL

)

Note that a default history table can be created automatically, using the script in #3, immediately below. You can create one manually, as here, but the history table must be identical to its ‘parent’ table in terms of its columns and datatypes. The history table cannot contain any triggers, unique or foreign keys, or constraints and cannot contain Filestream data. However, the history table could contain indexes, compression, or even a columnstore index.

1. Add the new columns to the table we want to make temporal for the period boundaries.

ALTER TABLE Person.BusinessEntityContact

ADD SysStartTime datetime2 GENERATED ALWAYS AS ROW START

CONSTRAINT P\_ValidFromConstraint DEFAULT SYSUTCDATETIME() NOT NULL,

SysEndTime datetime2 GENERATED ALWAYS AS ROW END

CONSTRAINT P\_ValidToConstraint DEFAULT CONVERT (DATETIME2, '9999-12-31 23:59:59.9999999') NOT NULL,

PERIOD FOR SYSTEM\_TIME (SysStartTime, SysEndTime)

1. Alter the non-temporal table to enable system-versioning and make a link to the new history table we created above.

ALTER TABLE Person.BusinessEntityContact

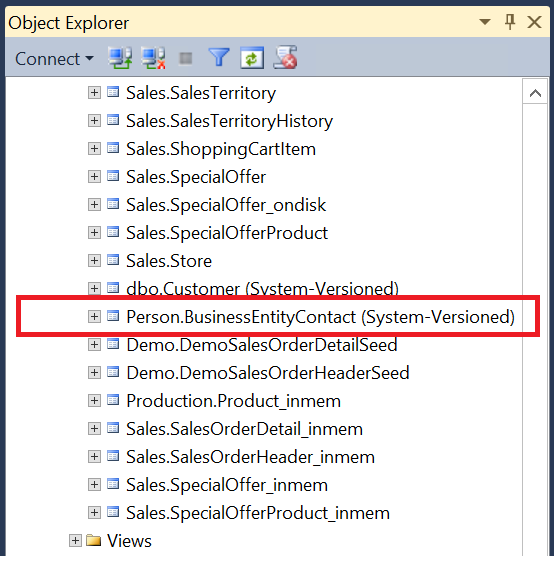
SET (SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = Person.BusinessEntityContactHistory))

|  |
| --- |
| Explore temporal data and metadata |

In this exercise, we will look at how temporal tables show up in SSMS and also take a look at the metadata for these tables.

### View temporal table in SSMS

1. In **Object Explorer**, refresh the tables by right clicking on **Tables** and selecting **Refresh**.
2. Notice that **Person.BusinessEntityContact** is now listed with “**(System-Versioned)**” after its name in Object Explorer and that it is listed below the non-temporal tables.



1. Also notice that the **Person.BusinessEntityContactHistory** table that we created above is not immediately visible in Object Explorer. To see the history tables, expand the plus sign next to the temporal table’s name. Note that the history table is listed “inside” its temporal parent table in Object Explorer.



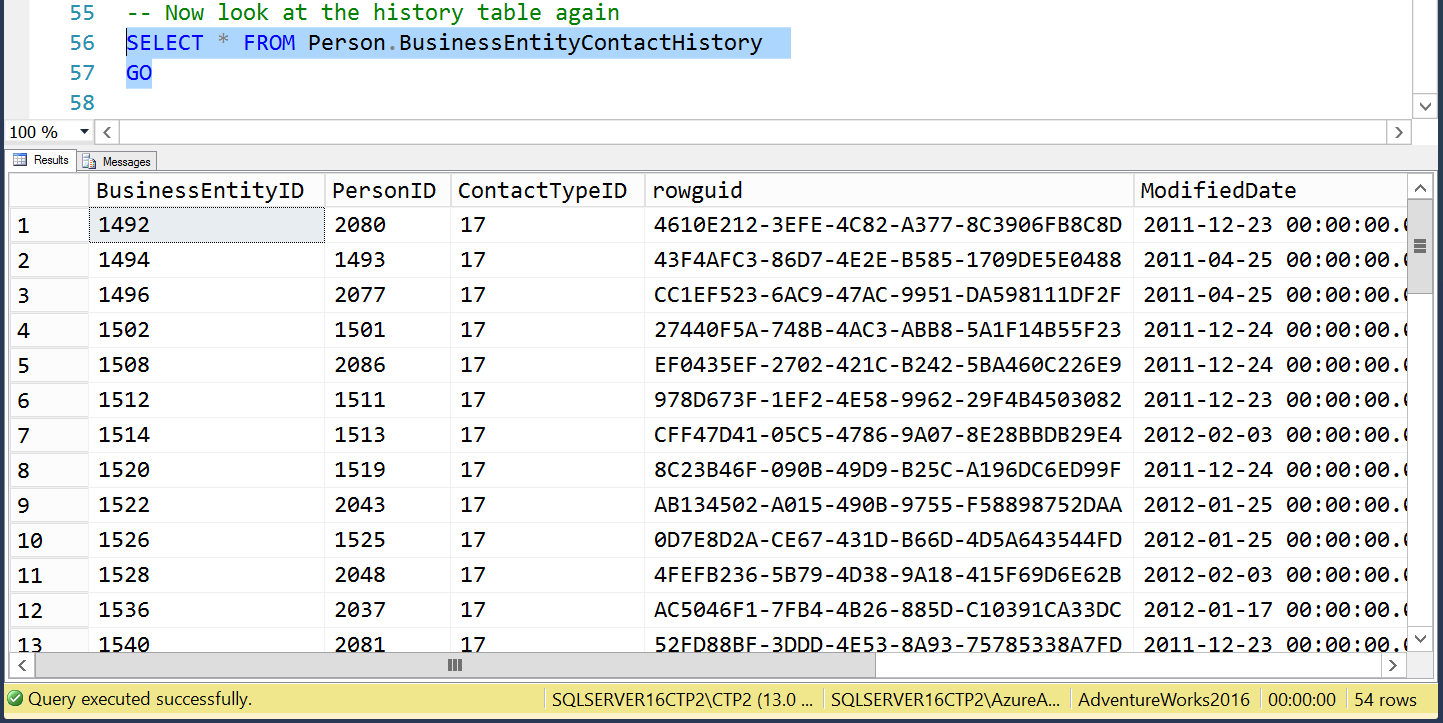
1. SELECT \* FROM Person.BusinessEntityContactHistory and note that there are not yet any records in the history table.
2. Update some rows.

UPDATE Person.BusinessEntityContact

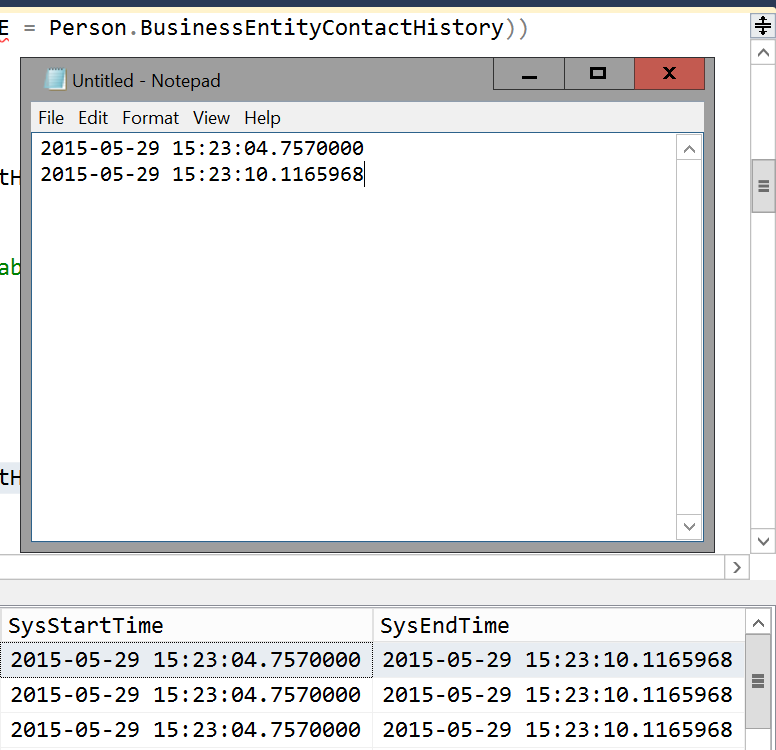
SET ContactTypeID = 19

WHERE ContactTypeID = 17

1. Now SELECT \* FROM Person.BusinessEntityContactHistory and notice that the pre-updated data is stored in the history table.



1. Copy the value from one of the rows for the SysStartTime and paste that UTC time value into Notepad.exe. Also copy the SysEndTime and paste on a new line in Notepad. We will use both times in a later part of this lab.



### View metadata about temporal tables

1. Execute the following query to view information about the temporal tables and their associated history tables.

SELECT T1.name as TemporalTableName, SCHEMA\_NAME(T1.schema\_id) AS TemporalTableSchema,

T2.name as HistoryTableName, SCHEMA\_NAME(T2.schema\_id) AS HistoryTableSchema,

T1.temporal\_type\_desc

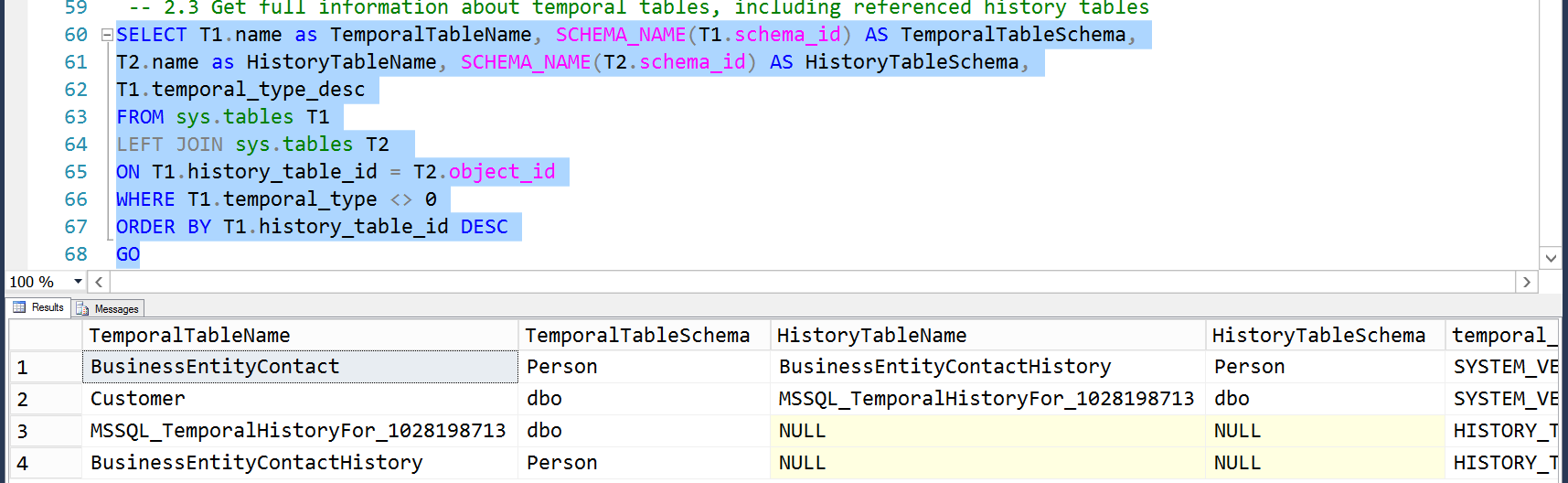
FROM sys.tables T1

LEFT JOIN sys.tables T2

ON T1.history\_table\_id = T2.object\_id

WHERE T1.temporal\_type <> 0

ORDER BY T1.history\_table\_id DESC



1. Execute the following query to view information about the Periods and their associated start and end time columns.

SELECT P.name as PeriodName, T.name as TemporalTableName, c1.name as StartPeriodColumnName, c2.name as EndPeriodColumnName

FROM sys.periods P

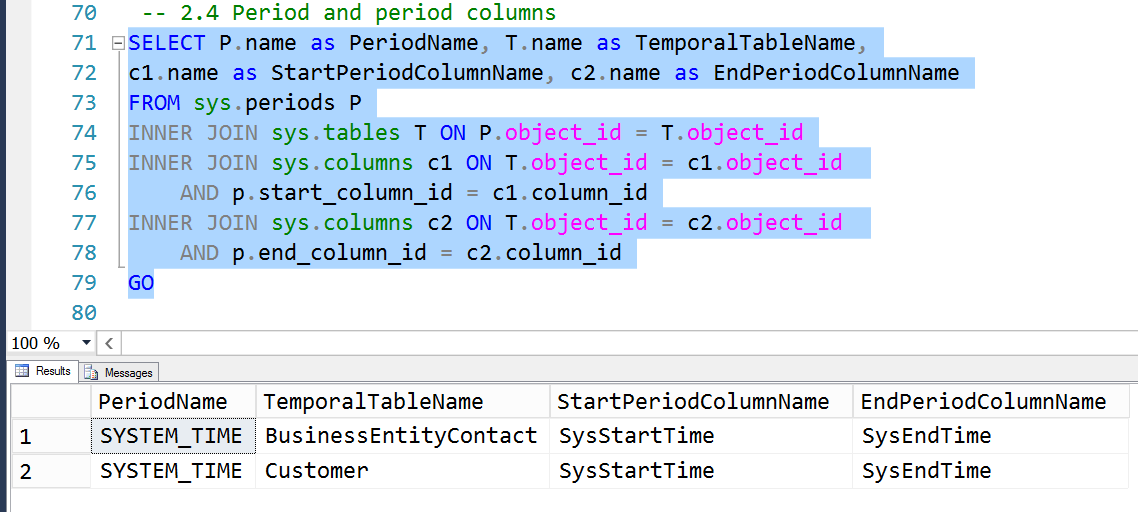
INNER JOIN sys.tables T ON P.object\_id = T.object\_id

INNER JOIN sys.columns c1 ON T.object\_id = c1.object\_id

AND p.start\_column\_id = c1.column\_id

INNER JOIN sys.columns c2 ON T.object\_id = c2.object\_id

AND p.end\_column\_id = c2.column\_id



|  |
| --- |
| Create temporal queries |

In this lab, we will explore how to query data using some of the temporal constructs. We’ll look at querying at a single point in time, as well as two ways to view records that span 2 points in time.

### Query the temporal table at a single point in time

This uses the AS OF syntax to query the table at a single point in time.

1. Run the following select statement. You should see zero rows, because we updated all **ContactTypesIDs** that were 17 so that now they are now 19, above.

SELECT \* FROM Person.BusinessEntityContact

WHERE ContactTypeID = 17

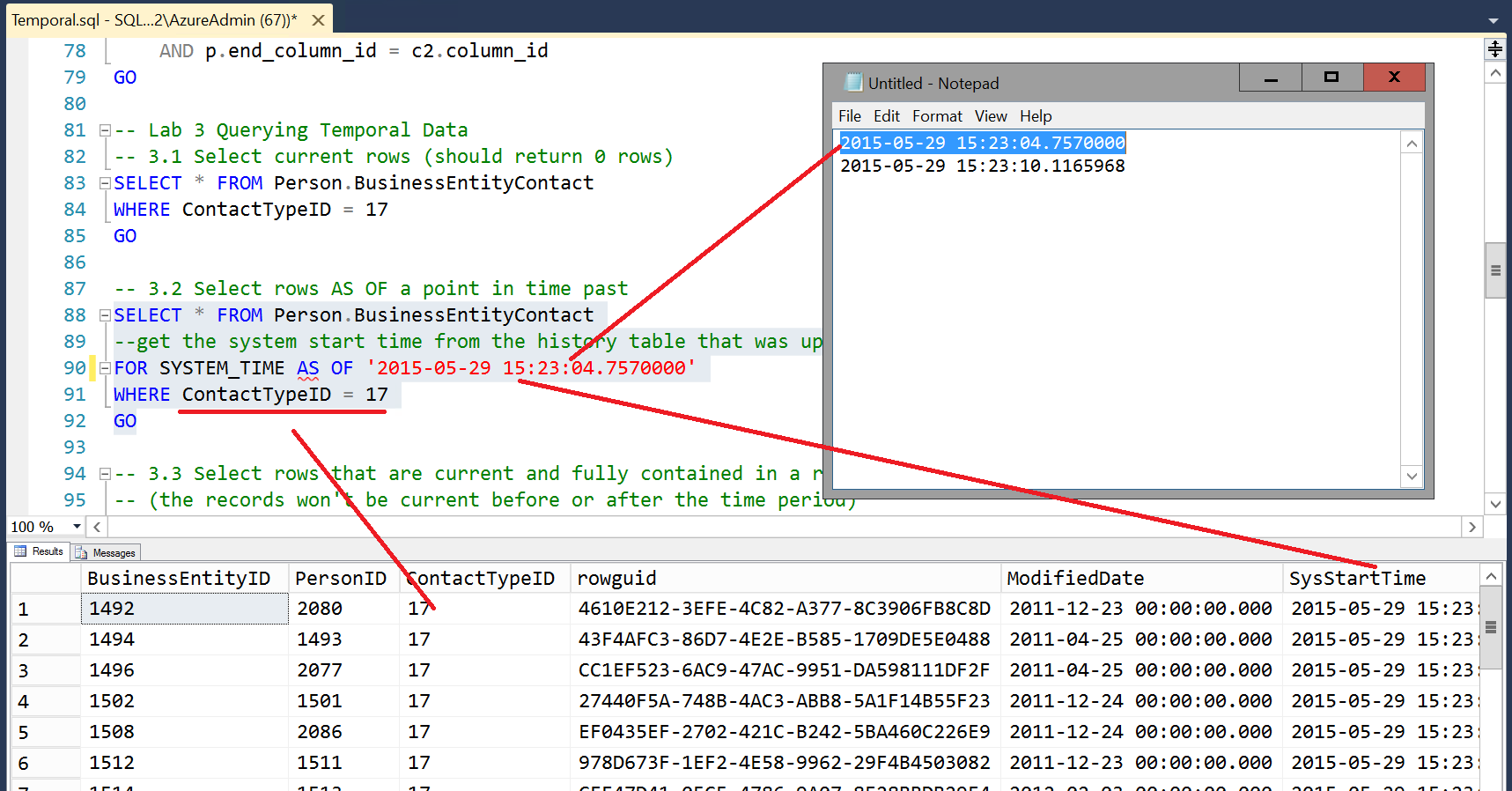
1. Now, we will run this same query again, but it will be as if we were running it before the update that we made to the ContactTypeIDs in the last part of the lab. Use the UTC start time value that you copied into Notepad in this query. It should return the rows with the ContactTypeID that had the value of 17 at that time.

SELECT \* FROM Person.BusinessEntityContact

--get the system start time from the history table that was saved to Notepad

FOR SYSTEM\_TIME AS OF '2015-04-06 19:16:28.4126016'

WHERE ContactTypeID = 17



Note: Using the construct FOR SYSTEM\_TIME AS OF @PointInTime will return rows where the @PointInTime is >= SysStartTime and < SysEndTime. If @PointInTime = SysEndTime those rows will not be returned. Make sure that the time you use is either equal to a start time or between the start and end time.

### Query the table for rows that were contained between 2 points in time

This uses CONTAINED IN to return only those rows that are contained within a period without overlapping the beginning or the end. The record version must be opened and closed within the period contained within the two points and cannot extend before or after that period.

1. First use the SysStartTime and SysEndTime that you copied to Notepad to define the period.

-- Get these 2 values from the SysStartTime and

--SysEndTime from the history table in 2.2 above

DECLARE @Start datetime2 = '2015-04-06 23:14:33.2370000'

DECLARE @End datetime2 = '2015-04-06 23:15:03.4098982'

SELECT \* FROM Person.BusinessEntityContact

FOR SYSTEM\_TIME CONTAINED IN(@Start, @End)

WHERE ContactTypeID = 17

1. Now modify the starting time and make it greater than it was originally, but still before the end time. Now, when you re-run the query, it will return no rows because the life of the rows with a ContactTypeID of 17 began before the period that we are querying for.

Note: You can think of CONTAINED IN as meaning that the full life of the rows as defined must have been ‘born’ and ‘died’ within the period. Any rows that were born before or that continued living after the period will not be returned.

### Query the table for rows that were current between 2 points in time, including overlaps or that immediately succeed them

The BETWEEN syntax returns table records that were active or began to be active within a given period of time. Record versions may be opened before start of period and stay opened after end of period. We’ll use the same scenario as for CONTAINED IN to see the difference

1. Again we will first use the SysStartTime and SysEndTime that you copied to Notepad to define the period.

-- Get these 2 values from the SysStartTime and

-- SysEndTime from the history table in 2.2 above

DECLARE @Start datetime2 = '2015-04-06 23:14:33.2370000'

DECLARE @End datetime2 = '2015-04-06 23:15:03.4098982'

SELECT \* FROM Person.BusinessEntityContact

FOR SYSTEM\_TIME BETWEEN @Start AND @End

WHERE ContactTypeID = 17

1. Again, just as for the CONTAINED IN scenario, modify the starting time and make it greater than it was originally, but still less than the end time. Now, however, when you re-run the query, it will still return the same rows because the life of the rows with a ContactTypeID of 17 is inside the period that we are querying for, even if that life began before the period or if it extends after the end of the period.

Note: The BETWEEN construct will also return rows that begin at the end of the period. In other words, those rows will be returned if their life immediately succeeds the defined period. There is an additional construct using FROM … TO that is identical to BETWEEN, except that FROM … TO does not include rows that immediately succeed the period.

### Query a view containing temporal tables

Creating a view that contains a temporal table is convenient for querying complex data that needs to be examined at various points in time.

1. Create a view that contains useful information related to the temporal table created earlier. Here we will look at information related to the business entity and their contacts.

CREATE VIEW Person.vw\_BEContactSummary

AS

SELECT P.BusinessEntityID AS PersonID, P.FirstName, P.LastName,

BEC.BusinessEntityID, CT.ContactTypeID, Name, A.AddressLine1, A.City, A.PostalCode

FROM [Person].[BusinessEntityContact] BEC

INNER JOIN [Person].[BusinessEntityAddress] BEA ON BEC.BusinessEntityID = BEA.BusinessEntityID

INNER JOIN [Person].[Address] A ON BEA.AddressID = A.AddressID

INNER JOIN [Person].[Person] P ON BEC.PersonID = P.BusinessEntityID

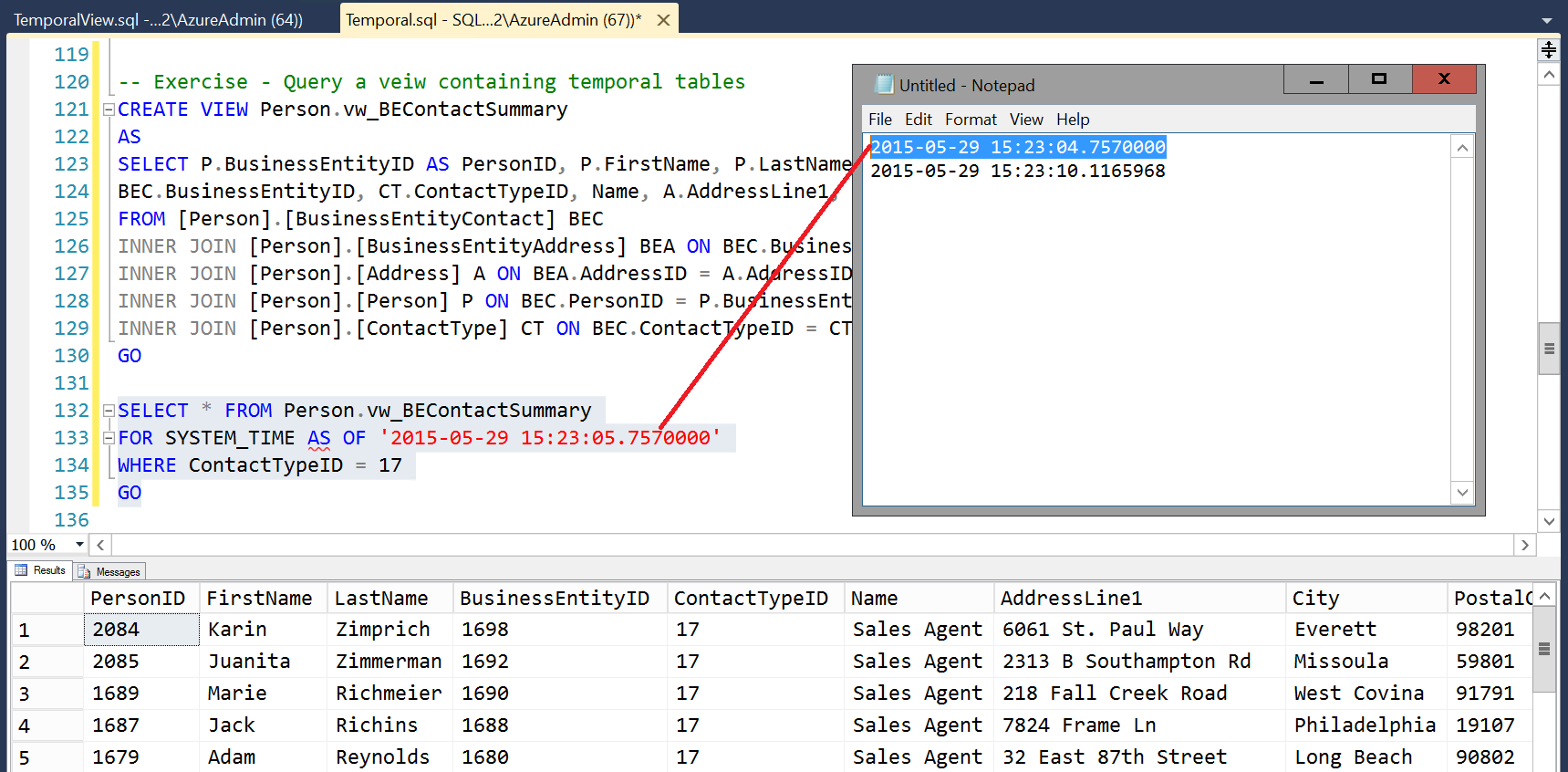
INNER JOIN [Person].[ContactType] CT ON BEC.ContactTypeID = CT.ContactTypeID

1. Now query that view for a specific SYSTEM\_TIME.

SELECT \* FROM Person.vw\_BEContactSummary

FOR SYSTEM\_TIME AS OF '2015-04-06 23:14:33.2370000'

WHERE ContactTypeID = 17



1. Change the value for the SYSTEM\_TIME to be a value after the end time that was copied into Notepad.exe and execute the query again. You should see no values.

|  |
| --- |
| Manage and maintain temporal indexes |

In this exercise, we will explore how to conduct maintenance tasks on the temporal table to make sure that performance is kept at the optimal level. Note that the current and the history components of the temporal table are independently maintained. Some of these tasks can be done as with other tables, but if maintenance operations would change the content of history data, it will be possible only while system versioning is suspended.

### Operations that can be run normally on temporal tables

1. Rebuild the indexes on both the current and history tables.

ALTER INDEX ALL ON Person.BusinessEntityContact REBUILD

GO

ALTER INDEX ALL ON Person.BusinessEntityContactHistory REBUILD

GO

1. Create statistics on the history table

CREATE STATISTICS [BusEntContact\_BusEnt]

ON Person.BusinessEntityContact([BusinessEntityID])

GO

1. Apply different compression on the history table

ALTER TABLE Person.BusinessEntityContact

REBUILD WITH (DATA\_COMPRESSION = PAGE)

### Operations that must be run with SYSTEM\_VERSIONING = OFF

Any operation that moves data, such as a partition change, or updates, inserts, or deletes data from a history table can only be accomplished with SYSTEM\_VERSIONING = OFF. Here, we’ll assume that the DBA wants to move older history into a new history archive table.

1. First create the new archive table.

SELECT TOP 0 \* INTO Person.BusinessEntityContactHistoryArchive

FROM Person.BusinessEntityContactHistory

1. Disable the temporal features on the main (current) table

ALTER TABLE Person.BusinessEntityContact

SET (SYSTEM\_VERSIONING = OFF)

1. Run any data moving or changing operations with the temporal features turned off.

INSERT INTO Person.BusinessEntityContactHistoryArchive

SELECT \* FROM Person.BusinessEntityContactHistory

DELETE FROM Person.BusinessEntityContactHistory

1. Enable the temporal features again and establish the link to the history table again.

ALTER TABLE Person.BusinessEntityContact

SET (SYSTEM\_VERSIONING = ON (HISTORY\_TABLE=Person.BusinessEntityContactHistory,

DATA\_CONSISTENCY\_CHECK = OFF))

1. You can now close the lab environment.

|  |
| --- |
| Terms of use |

© 2015 Microsoft Corporation. All rights reserved.

By using this Hands-on Lab, you agree to the following terms:

The technology/functionality described in this Hands-on Lab is provided by Microsoft Corporation in a “sandbox” testing environment for purposes of obtaining your feedback and to provide you with a learning experience. You may only use the Hands-on Lab to evaluate such technology features and functionality and provide feedback to Microsoft.  You may not use it for any other purpose. You may not modify, copy, distribute, transmit, display, perform, reproduce, publish, license, create derivative works from, transfer, or sell this Hands-on Lab or any portion thereof.

COPYING OR REPRODUCTION OF THE HANDS-ON LAB (OR ANY PORTION OF IT) TO ANY OTHER SERVER OR LOCATION FOR FURTHER REPRODUCTION OR REDISTRIBUTION IS EXPRESSLY PROHIBITED.

THIS HANDS-ONLAB PROVIDES CERTAIN SOFTWARE TECHNOLOGY/PRODUCT FEATURES AND FUNCTIONALITY, INCLUDING POTENTIAL NEW FEATURES AND CONCEPTS, IN A SIMULATED ENVIRONMENT WITHOUT COMPLEX SET-UP OR INSTALLATION FOR THE PURPOSE DESCRIBED ABOVE.  THE TECHNOLOGY/CONCEPTS REPRESENTED IN THIS HANDS-ON LAB MAY NOT REPRESENT FULL FEATURE FUNCTIONALITY AND MAY NOT WORK THE WAY A FINAL VERSION MAY WORK.  WE ALSO MAY NOT RELEASE A FINAL VERSION OF SUCH FEATURES OR CONCEPTS.  YOUR EXPERIENCE WITH USING SUCH FEATURES AND FUNCITONALITY IN A PHYSICAL ENVIRONMENT MAY ALSO BE DIFFERENT.

**FEEDBACK**.  If you give feedback about the technology features, functionality and/or concepts described in this Hands-on Lab to Microsoft, you give to Microsoft, without charge, the right to use, share and commercialize your feedback in any way and for any purpose.  You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft software or service that includes the feedback.  You will not give feedback that is subject to a license that requires Microsoft to license its software or documentation to third parties because we include your feedback in them.  These rights survive this agreement.

MICROSOFT CORPORATION HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH REGARD TO THE HANDS-ON LAB , INCLUDING ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT.  MICROSOFT DOES NOT MAKE ANY ASSURANCES OR REPRESENTATIONS WITH REGARD TO THE ACCURACY OF THE RESULTS, OUTPUT THAT DERIVES FROM USE OF THE VIRTUAL LAB, OR SUITABILITY OF THE INFORMATION CONTAINED IN THE VIRTUAL LAB FOR ANY PURPOSE.

DISCLAIMER

This lab contains only a portion of new features and enhancements in Microsoft SQL Server 2016 CTP2. Some of the features might change in future releases of the product. In this lab, you will learn about some, but not all, new features.