# Intro

In this document, I detail how to take data from an Azure SQL MI database and copy it to a dedicated pool using a combination of the following two articles and some additional steps.

[Incrementally copy data using Change Data Capture - Azure Data Factory | Microsoft Docs](https://docs.microsoft.com/en-us/azure/data-factory/tutorial-incremental-copy-change-data-capture-feature-portal)

[Incrementally copy data from a source data store to a destination data store - Azure Data Factory | Microsoft Docs](https://docs.microsoft.com/en-us/azure/data-factory/tutorial-incremental-copy-overview)

Table of Contents

[Intro 1](#_Toc86256691)

[Prerequisites 1](#_Toc86256692)

[Creating the incremental Copy Pipeline 3](#_Toc86256693)

[Create the PipelineReference database 3](#_Toc86256694)

[Creating the pipeline 4](#_Toc86256695)

[Creating the Trigger SP Pipeline 14](#_Toc86256696)

[Preparing the Tables 14](#_Toc86256697)

[Creating the Stored Procedures 14](#_Toc86256698)

[Creating the pipeline 20](#_Toc86256699)

# Prerequisites

Steps taken to replicate the initial environment…

1. We first create an AdventureWorksLT database in our Azure SQL MI instance. This can be achieved by utilizing the Adventure Works sample databases at <https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver15&tabs=ssms>
2. Next we need to create a blank AdventureWorks Dedicated SQL Pool on our Azure Synapse Instance.
3. Within our AdventureWorks SQL Dedicated Pool we need to add the staging schema to the database. Within the Synapse Workspace, create a new SQL script under Develop. Point it to the SQL Dedicated Pool and copy the code below and run it.

CREATE SCHEMA SalesLT;

CREATE SCHEMA Staging

1. We then copy the tables in the AdventureWorksLT database to our AdventureWorks Synapse Dedicated Pool using the Built-In copy task within Synapse… [Copy Data tool - Azure Data Factory & Azure Synapse | Microsoft Docs](https://docs.microsoft.com/en-us/azure/data-factory/copy-data-tool?tabs=data-factory)

NOTE: This created all tables in the Dedicated Pool as ROUND ROBIN by Default. You may need to change these tables to Hash Distributed. You can do that using the CTAS command. For more on these topics check out the following.

[Distributed tables design guidance - Azure Synapse Analytics | Microsoft Docs](https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-tables-distribute)

[CREATE TABLE AS SELECT (CTAS) - Azure Synapse Analytics | Microsoft Docs](https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-develop-ctas)

1. Now that we have 2 databases the are essentially the same, we will turn on Change Data Capture within the AdventureWorksLT SQL Azure Instance to track all inserts, updates and deletes so that they can be utilized in the incremental pipeline. For more on Data Capture check out [What is change data capture (CDC)? - SQL Server | Microsoft Docs](https://docs.microsoft.com/en-us/sql/relational-databases/track-changes/about-change-data-capture-sql-server?view=sql-server-ver15)

# Creating the incremental Copy Pipeline

We did not fully utilize the following instructions [Incrementally copy data using Change Data Capture - Azure Data Factory | Microsoft Docs](https://docs.microsoft.com/en-us/azure/data-factory/tutorial-incremental-copy-change-data-capture-feature-portal) as we had some differences and also other factors to track.

1. This assumes that the time window trigger is always running as expected and if the trigger fails or is accidentally deactivated, you will need to adjust your query to account for the larger time window. Because of that, I prefer using Watermarking. The basics of using watermarking with pipelines is defined here… [Incrementally copy data from a source data store to a destination data store - Azure Data Factory | Microsoft Docs](https://docs.microsoft.com/en-us/azure/data-factory/tutorial-incremental-copy-overview)
2. This accounts for writing to Azure Storage and we want to write to a dedicated SQL pool but we are writing to the staging tables of the dedicated pool. We will use a combination of Stored Procedures to update the production tables from the staging tables in the dedicated pool.

## Create the PipelineReference database

Thus we are going to perform a few steps to create an incremental copy job. For this exercise we will just focus on the SalesLT.Address and SalesLT.Customer tables in the AdventureWorksLT Azure SQL instance. We used the following steps to get started.

1. To begin we need to create another DB on our Azure SQL instance. I named the database **PipelineReference** and populated it with the two tables using the following scripts
   1. Tablelookup: This is used to designate which tables will be copied from the SQL MI instance to the Dedicated Pool.  
      Use the script below to create the table

CREATE TABLE [dbo].[tablelookup](

[SourceTableSchema] [nvarchar](25) NOT NULL,

[SourceTableName] [nvarchar](100) NOT NULL,

[SourceCDCTableName] [nvarchar](150) NOT NULL,

[TargetTableSchema] [nvarchar](25) NOT NULL,

[TargetTableName] [nvarchar](100) NOT NULL

) ON [PRIMARY]

GO

You can also use the following script to populate the table.   
  
INSERT INTO [dbo].[tablelookup]

([SourceTableSchema]

,[SourceTableName]

,[SourceCDCTableName]

,[TargetTableSchema]

,[TargetTableName])

VALUES

('SalesLT', 'Address', 'SalesLT\_Address\_CT', 'Staging', 'Address'),

('SalesLT', 'Customer', 'SalesLT\_Customer\_CT', 'Staging', 'Customer')

GO

* 1. Watermark table. We will use this to store our values needed for the incremental copy.   
     Use the script below to create the table

CREATE TABLE [dbo].[watermarktable](

[SourceCDCTableName] [varchar](150) NULL,

[WatermarkValue] [datetime] NULL

) ON [PRIMARY]

GO

You can use the following script to populate the table.

INSERT INTO [dbo].[watermarktable]

([SourceCDCTableName]

,[WatermarkValue])

VALUES

('SalesLT\_Address\_CT', '2021-10-22 00:00:00.000'),

('SalesLT\_Customer\_CT', '2021-10-22 00:00:00.000')

1. Next we need to add the following stored procedure to the **PipelineReference** database.  
     
   CREATE PROCEDURE [dbo].[usp\_write\_watermark] @LastModifiedtime datetime, @TableName varchar(50)

AS

BEGIN

UPDATE watermarktable

SET [WatermarkValue] = @LastModifiedtime

WHERE [SourceCDCTableName] = @TableName

END

GO

## Creating the pipeline

1. Now we will create an initial pipeline with a lookup against our new tablelookup table and then execute a child pipeline based on the results.  
   Diagram

   Description automatically generated with low confidence
   1. Create a pipeline and name it **GetTableList-Trigger-Incremental-Copy**
   2. On the settings tab of the Lookup Activity and we want to set the following…
      1. Source Dataset – Point to the **PipelineReference** Azure SQL database
      2. Use Query: Query
      3. Query: select \* from [dbo].[tablelookup]

Graphical user interface, text, application, email

Description automatically generated

* 1. Publish this pipeline, we will add the Execute Pipeline step later.

1. Now we will create our child pipeline that will take the lookup results from the PipelineReference database and loop through each set of tables to incrementally copy the changes to the staging tables of the Dedicated SQL Pool. Below are the various steps of the pipeline that we will create.

Graphical user interface, application

Description automatically generated

Diagram

Description automatically generated with medium confidence

A picture containing waterfall chart

Description automatically generated

1. Create a new pipeline and name it **Iterate-and-Copy-SQL-Tables**.
2. In the Parameters tab of the pipeline add a New Parameter with the following values:
   1. Name = tableList
   2. Type=Array
   3. Default Value should be blank. When blank you will see “Value”

Graphical user interface, text, application, email

Description automatically generated

1. Add a ForEach activity to the Pipeline canvas and name it **IterateSQLTables**
   1. On the settings tab, set Items = @pipeline().parameters.tableList  
      Graphical user interface, text, application, email

      Description automatically generated
2. In the ForEach control on the Activities tab, click the pencil to open the canvas specific to our ForEach activity. This is where we will write most of the logic.
3. Add Three Lookup acitivites to the canvas of the **IterateSQLTables** ForEach activity.
   1. For the first Lookup, name it **LookupOldWaterMarkActivity**.

Under Settings, we need to create a dataset that links to the Watermark table in the **PipelineReference** database on the Azure SQL instance that we created.  
NOTE: This will also involve creating a Linked Service to the Azure SQL instance and then linking the dataset to that Linked Service.

* + 1. Once created, set the new dataset equal to the PipelineReference dataset.
    2. Use query: query
    3. Query: select \* from [dbo].[watermarktable] where [SourceCDCTableName] = '@{item().SourceCDCTableName}'  
       **NOTE**: This captures the last record updated in the previous pipeline run and will be used as our start date when querying for changes.
    4. First row only: Checked  
       Graphical user interface, text, application, email

       Description automatically generated
  1. For the second Lookup, name it **LookupNewWaterMarkActivity**.  
     Under Settings, we need to create a new dataset that links to the AdventureWorksLT source database on the Azure SQL Instance  
     NOTE: This will also involve creating a Linked Service to the AdventureWorksLT Azure SQL instance and then linking the dataset to that Linked Service.
     1. Once created, set the new dataset equal to the AdventureWOorksLT dataset.
     2. Use query: query
     3. Query: select MAX(TM.tran\_begin\_time) as NewWatermarkValue from [cdc].[@{item().SourceCDCTableName}] CDC INNER JOIN [cdc].[lsn\_time\_mapping] TM on CDC.\_\_$start\_lsn = TM.start\_lsn  
        **NOTE**: We are using this query instead of the function utilized in the CDC Capture Incremental Copy Pipeline article referenced earlier. This query is based on the following…

select MAX(TM.tran\_begin\_time) as NewWatermarkValue from [cdc].[SalesLT\_Customer\_CT] CDC INNER JOIN [cdc].[lsn\_time\_mapping] TM on CDC.\_\_$start\_lsn = TM.start\_lsn  
  
This captures the most recent datetime field when the last cdc event was captured and we can use this to query for changed records as our end time.

* + 1. First row only: Checked  
       Graphical user interface, text, application, email

       Description automatically generated
  1. For the third Lookup, name it **GetChangeCount**.
     1. Once created, set the new dataset equal to the dataset you used for the **LookupNewWaterMarkActivity** activity.
     2. Use query: query
     3. Query: select count(\*) as changecount from [cdc].[@{item().SourceCDCTableName}] CDC INNER JOIN [cdc].[lsn\_time\_mapping] TM on CDC.\_\_$start\_lsn = TM.start\_lsn

where tran\_begin\_time >

'@{activity('LookupOldWaterMarkActivity').output.firstRow.WatermarkValue}' and tran\_begin\_time <= '@{activity('LookupNewWaterMarkActivity').output.firstRow.NewWatermarkvalue}'  
**NOTE**: We are using this query to determine if there are any new records residing in the CDC tables of the Azure SQL instance since the last time we ran this pipeline. If none exist, we will skip that table. This will be defined in the **If** **Condition** activity that we will build in the next steps.

* + 1. First row only: Checked  
       Graphical user interface, text, application, email

       Description automatically generated

1. Next arrange and link the lookups so that the **LookupOldWaterMarkActivity and LookupNewWaterMarkActivity** outputs feed into the **GetChangeCount** lookup like the picture below.  
   Graphical user interface, text, application

   Description automatically generated
2. Add an **If Condition** activity to the canvas to the right of **GetChangeCount** and name it **HasChangedRows**.
   * 1. Link the output of **GetChangeCount** activity to the input of **HasChangedRows.**
     2. On the Acitivites tab of the HasChangedRows activity, set the expression to  
        @greater(int(activity('GetChangeCount').output.firstRow.changecount),0)  
        **NOTE**: This will trigger a true or false based on the row count. If we have data in the CDC tables newer than the old water mark, it will trigger a true activity.
3. Click the pencil on the True Activity Listing of the Activities tab to activate a new canvas under the **HasChangedRows** activity. Up top, your breadcrumbs should look like the following.  
   
4. Now add a Copy data activity to the canvas and set the following values
   1. Source Tab
      1. Set the Source Dataset to the same dataset you used for the **LookupNewWaterMarkActivity** and **GetChangedRows** activities.
      2. Use query: Query
      3. Query: select CDC.\*, TM.tran\_begin\_time from [cdc].[@{item().SourceCDCTableName}] CDC INNER JOIN [cdc].[lsn\_time\_mapping] TM on CDC.\_\_$start\_lsn = TM.start\_lsn

where tran\_begin\_time >

'@{activity('LookupOldWaterMarkActivity').output.firstRow.WatermarkValue}' and tran\_begin\_time <= '@{activity('LookupNewWaterMarkActivity').output.firstRow.NewWatermarkvalue}'  
**NOTE**: We are using this query to get all values for any new records residing in the CDC tables of the Azure SQL instance since the last time we ran this pipeline.   
Graphical user interface, text, application, email

Description automatically generated

* 1. Sink Tab
     1. For Sink dataset, we need to create a new dataset that is linked to our Dedicated SQL Pool with the following values
        1. Connection Tab
           1. SQL pool: AdventureWorks  
              NOTE: This is our target database
           2. Table: @dataset()[.DWTableSchema.@dataset().DWTableName](mailto:.DWTableSchema.@dataset().DWTableName)
           3. Edit: Checked.

Graphical user interface, application

Description automatically generated

* + - 1. Parameters Tab  
         Create two parameters with the following values.

|  |  |  |
| --- | --- | --- |
| Name | Type | Default Value |
| DWTableSchema | String | Leave Blank |
| DWTableName | String | Leave Blank |

Graphical user interface, application

Description automatically generated

* + 1. Set this new dataset to your sink dataset value on the Sink tab of the True activities canvas
    2. Under the Dataset properties set the following values
       1. DWTableSchema: @{item().TargetTableSchema}
       2. DWTableName: @{item().TargetTableName}
    3. Copy Method: Bulk Insert  
       NOTE: For this example we are using Bulk Insert to auto create the tables. In production scenarios we would recommend that after the tables are created to use either COPY or Polybase which will requires Staging data in a storage account. For more on this topic see  
       [Copy and transform data in Azure Synapse Analytics - Azure Data Factory & Azure Synapse | Microsoft Docs](https://docs.microsoft.com/en-us/azure/data-factory/connector-azure-sql-data-warehouse?tabs=data-factory#azure-sql-data-warehouse-as-sink)
    4. Table option: Auto create table

NOTE: This will create the table with a Round Robin Distribution

Graphical user interface, text, application, email

Description automatically generated

1. Next add a Stored Procedure Activity to the canvas to the right of the Copy data activity and name it **WriteWatermarkActivity**. Link the output of the Copy data activity to the input of the **WriteWatermarkActivity** so that it looks like the screenshot below.  
   A picture containing diagram

   Description automatically generated
   1. Under the Settings tab of the **WriteWatermarkActivity** set the following values:
      1. Linked Service: This is tied to the **PipelineReference** database of your Azure SQL Instance.
      2. Stored procedure name: [dbo].[usp\_write\_watermark]
      3. Edit: Checked
         1. Add 2 Stored Procedure Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| LastModifiedtime | Datetime | @{activity('LookupNewWaterMarkActivity').output.firstRow.NewWatermarkvalue} |
| TableName | String | @{activity('LookupOldWaterMarkActivity').output.firstRow.SourceCDCTableName} |

Graphical user interface, text, application, email

Description automatically generated

1. Next we want to click back to the beginning of the breadcrumbs so we are at the root of the **Iterate-and-Copy-SQL-Tables** pipeline.  
   
2. Now reopen the **GetTableList-Trigger-Incremental-Copy** pipeline and add an Execute Pipeline activity to the right of the **LookupTableList** activity and name it **TriggerCopy**.
   1. On the Settings tab, set the following values
      1. Invoked Pipeline: Iterate-and-Copy-SQL-Tables
      2. Wait on completion: Checked
      3. Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| tableList | array | @activity('LookupTableList').output.value |

Graphical user interface, text, application

Description automatically generated

1. Link the output of the **LookupTableList** activity to the input of the TriggerCopy so the canvas looks like the following.  
   A picture containing waterfall chart

   Description automatically generated
2. Publish your pipelines, datasets, etc.
3. Now we can test this by adding, updating and deleting records from the SalesLT.Address and SalesLT.Customer tables in the AdventureWorksLT SQL instance.
4. When you do the previous step, it will create new rows under the system tables of [cdc].[SalesLT\_Address\_CT] and [cdc].[SalesLT\_Contact\_CT] tables. To understand the values in these tables check out  
   [What is change data capture (CDC)? - SQL Server | Microsoft Docs](https://docs.microsoft.com/en-us/sql/relational-databases/track-changes/about-change-data-capture-sql-server?view=sql-server-ver15#change-table)
5. Open the **GetTableList-Trigger-Incremental-Copy** and trigger the pipeline. When complete you should now have two new tables in your dedicated pool named [Staging].[Address] and [Staging].[customer].
6. You will also notice that the watermark values have been updated in the PipelineReference Azure SQL instance. To evaluate these run the following SQL script.  
   select \* from [dbo].[watermarktable]

**In the next section we will create the stored procedures and pipeline to copy the data from the staging tables to the production tables within the SQL Dedicated Pool.**

# Creating the Trigger SP Pipeline

## Preparing the Tables

In preparation for the subsequent steps, we need to add 2 fields to each of the production tables within the SQL Dedicated Pool. These are **\_\_$operation** and **tran\_begin\_time** and these will be used to evaluate rows in advanced delete and other scenarios. Here are commands for the Address and Contact tables.

ALTER TABLE [SalesLT].[Address]

ADD [\_\_$operation] [int] NULL,

[tran\_begin\_time] [datetime] NULL;

ALTER TABLE [SalesLT].[Customer]

ADD [\_\_$operation] [int] NULL,

[tran\_begin\_time] [datetime] NULL;

## Creating the Stored Procedures

These will be utilized in the subsequent pipeline we create. Because this could be set on a rolling window, a record could be added, updated and deleted in the source Azure SQL database during that window. Thus, we will create 3 stored procedures per entity:

1. Insert
2. Update
3. Delete

These are fired by one master Stored Procedure which we will trigger from the pipeline created later in this section.

1. Create the following spInsert Stored Procedures in the AdventureWorks SQL Dedicated Pool.
   1. SalesLT.spInsertAddress  
      IF OBJECT\_ID ( 'SalesLT.spInsertAddress', 'P' ) IS NOT NULL

    DROP PROCEDURE SalesLT.spInsertAddress;

GO

CREATE PROCEDURE SalesLT.spInsertAddress

AS

BEGIN

    SET NOCOUNT ON;

    INSERT SalesLT.Address

           (

                AddressID

                , AddressLine1

                , AddressLine2

                , City

                , StateProvince

                , CountryRegion

                , PostalCode

                , rowguid

                , ModifiedDate

              , \_\_$operation

              , tran\_begin\_time

           )

    SELECT

                  AddressID

                , AddressLine1

                , AddressLine2

                , City

                , StateProvince

                , CountryRegion

                , PostalCode

                , rowguid

                , ModifiedDate

              , \_\_$operation

              , tran\_begin\_time

                from Staging.Address S1

                where NOT EXISTS (SELECT AddressID FROM SalesLT.Address S2 WHERE S2.AddressID = S1.AddressID)

                and S1.\_\_$operation = 2

;

END;

GO

* 1. SalesLT.spInsertCustomer

IF OBJECT\_ID ( 'SalesLT.spInsertCustomer', 'P' ) IS NOT NULL

    DROP PROCEDURE SalesLT.spInsertCustomer;

GO

CREATE PROCEDURE SalesLT.spInsertCustomer

AS

BEGIN

    SET NOCOUNT ON;

    INSERT SalesLT.Customer

           (

                CustomerID

                , NameStyle

                , Title

                , FirstName

                , MiddleName

                , LastName

                , Suffix

                , CompanyName

                , SalesPerson

                , EmailAddress

                , Phone

                , PasswordHash

                , PasswordSalt

                , rowguid

                , ModifiedDate

                , \_\_$operation

                , tran\_begin\_time

           )

    SELECT

                CustomerID

                , NameStyle

                , Title

                , FirstName

                , MiddleName

                , LastName

                , Suffix

                , CompanyName

                , SalesPerson

                , EmailAddress

                , Phone

                , PasswordHash

                , PasswordSalt

                , rowguid

                , ModifiedDate

                , \_\_$operation

                , tran\_begin\_time

                from Staging.Customer S1

                where NOT EXISTS (SELECT CustomerID FROM SalesLT.Customer S2 WHERE S2.CustomerID = S1.CustomerID)

                and S1.\_\_$operation = 2

;

END;

GO

1. Create the following spUpdate Stored Procedures in the AdventureWorks SQL Dedicated Pool.
   1. SalesLT.spUpdateAddress

IF OBJECT\_ID ( 'SalesLT.spUpdateAddress', 'P' ) IS NOT NULL

    DROP PROCEDURE SalesLT.spUpdateAddress;

GO

CREATE PROCEDURE SalesLT.spUpdateAddress

AS

BEGIN

    SET NOCOUNT ON;

    UPDATE SalesLT.Address

           SET

                AddressID = S1.AddressID

                , AddressLine1 = S1.AddressLine1

                , AddressLine2 = S1.AddressLine2

                , City = S1.City

                , StateProvince = S1.StateProvince

                , CountryRegion = S1.CountryRegion

                , PostalCode = S1.PostalCode

                , rowguid = S1.rowguid

                , ModifiedDate = S1.ModifiedDate

              , \_\_$operation = S1.\_\_$operation

              , tran\_begin\_time = S1.tran\_begin\_time

                from Staging.Address S1

                where [SalesLT].[Address].AddressID = S1.AddressID

                and S1.\_\_$operation = 4

                and S1.tran\_begin\_time = (select MAX(s2.tran\_begin\_time) from Staging.Address s2 where s2.addressid=s1.addressid and S2.\_\_$operation in (4))

;

END;

GO  

* 1. SalesLT.spUpdateCustomer

IF OBJECT\_ID ( 'SalesLT.spUpdateCustomer', 'P' ) IS NOT NULL

    DROP PROCEDURE SalesLT.spUpdateCustomer;

GO

CREATE PROCEDURE SalesLT.spUpdateCustomer

AS

BEGIN

    SET NOCOUNT ON;

    UPDATE SalesLT.Customer

           SET

                NameStyle = S1.NameStyle

                , Title = S1.Title

                , FirstName = S1.FirstName

                , MiddleName = S1.MiddleName

                , LastName = S1.LastName

                , Suffix = S1.Suffix

                , CompanyName = S1.CompanyName

                , SalesPerson = S1.SalesPerson

                , EmailAddress = S1.EmailAddress

                , Phone = S1.Phone

                , PasswordHash= S1.PasswordHash

                , PasswordSalt = S1.PasswordSalt

                , rowguid = S1.rowguid

                , ModifiedDate = S1.ModifiedDate

                , \_\_$operation = S1.\_\_$operation

                , tran\_begin\_time = S1.tran\_begin\_time

                from Staging.Customer S1

                where [SalesLT].[Customer].CustomerID = S1.CustomerID

                and S1.\_\_$operation = 4

                and S1.tran\_begin\_time = (select MAX(s2.tran\_begin\_time) from Staging.Customer s2 where s2.Customerid=s1.Customerid and S2.\_\_$operation in (4))

;

END;

GO

1. Create the following spDelete Stored Procedures in the AdventureWorks SQL Dedicated Pool.
   1. SalesLT.spDeleteAddress

IF OBJECT\_ID ( 'SalesLT.spDeleteAddress', 'P' ) IS NOT NULL

    DROP PROCEDURE SalesLT.spDeleteAddress;

GO

CREATE PROCEDURE SalesLT.spDeleteAddress

AS

BEGIN

    SET NOCOUNT ON;

DELETE FROM S2

FROM SalesLT.Address S2

    INNER JOIN [Staging].[Address] S1

           ON S2.AddressID = S1.AddressID

    Where S2.AddressID = S1.AddressID

    and S1.\_\_$operation = 1

;

END;

GO

* 1. SalesLT.spDeleteCustomer

IF OBJECT\_ID ( 'SalesLT.spDeleteCustomer', 'P' ) IS NOT NULL

    DROP PROCEDURE SalesLT.spDeleteCustomer;

GO

CREATE PROCEDURE SalesLT.spDeleteCustomer

AS

BEGIN

    SET NOCOUNT ON;

DELETE FROM S2

FROM SalesLT.Customer S2

    INNER JOIN [Staging].[Customer] S1

           ON S2.CustomerID = S1.CustomerID

    Where S2.CustomerID = S1.CustomerID

    and S1.\_\_$operation = 1

;

END;

GO

1. Last of all we need to create the master Stored Procedure that will use Dynamic SQL to execute the SPs we just created in the previous steps. This master stored procedure will be triggered by the pipeline we are going to create in the subsequent steps.

IF OBJECT\_ID ( 'SalesLT.spInsertUpdateDelete', 'P' ) IS NOT NULL

    DROP PROCEDURE SalesLT.spInsertUpdateDelete;

GO

CREATE PROC [SalesLT].[spInsertUpdateDelete]

(

    @TableSchema VARCHAR(50),

    @TableName VARCHAR(50)

)

 AS

DECLARE @sqlInsert AS NVARCHAR(4000)

DECLARE @sqlUpdate AS NVARCHAR(4000)

DECLARE @sqlDelete AS NVARCHAR(4000)

DECLARE @sqlTruncate AS NVARCHAR(4000)

SELECT @sqlInsert = 'EXEC ' + @TableSchema + '.spInsert' + @TableName + ''

EXEC(@sqlInsert)

SELECT @sqlUpdate = 'EXEC ' + @TableSchema + '.spUpdate' + @TableName + ''

EXEC(@sqlUpdate)

SELECT @sqlDelete = 'EXEC ' + @TableSchema + '.spDelete' + @TableName + ''

EXEC(@sqlDelete)

SELECT @sqlTruncate = 'TRUNCATE TABLE Staging.' + @TableName + ''

EXEC(@sqlTruncate)

GO

## Creating the pipeline

Now that we have the stored procedures created, we can create the Synapse Pipeline to execute them on an automated basis. The purpose of the pipeline is to scroll through the staging tables defined in the dbo.tablelookup table of the PipelineReference database. If we find data in the staging tables, the stored procedure will perform the following actions in the following order…

1. Insert Records with \_\_$operation = 2
2. Update Records with \_\_$operation = 4.   
   **NOTE:** We will take only the max record for each table grouped by unique ID and MAX(tran\_begin\_time).
3. Delete Records with \_\_$operation = 1
4. Truncate the staging table
5. Now we will create an initial pipeline with a lookup against the tablelookup table in the PipelineReference Azure SQL database and then execute a child pipeline based on the results.  
   A picture containing diagram

   Description automatically generated
   1. Create a pipeline and name it **GetTableList-Trigger-sp**
   2. On the settings tab of the Lookup Activity and we want to set the following…
      1. Source Dataset – Point to the **PipelineReference** Azure SQL database
      2. Use Query: Query
      3. Query: select \* from [dbo].[tablelookup]

Graphical user interface, text, application, email

Description automatically generated

* 1. Publish this pipeline, we will add the Execute Pipeline step later.

1. Now we will create our child pipeline that will take the lookup results and loop through each set of tables to incrementally copy the changes to the staging tables of the dedicated pool. Below are the various steps of the pipeline that we will create.

Graphical user interface, application, Teams

Description automatically generated

Diagram

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

1. Create a new pipeline and name it **Iterate-and-EXEC-sp**.
2. In the Parameters tab of the pipeline add a New Parameter with the following values:
   1. Name = tableList
   2. Type=Array
   3. Default Value should be blank. When blank you will see “Value”

Graphical user interface, text, application, email

Description automatically generated

1. Add a ForEach activity to the Pipeline canvas and name it **IterateSQLTables**
   1. On the settings tab, set Items = @pipeline().parameters.tableList  
      Graphical user interface, text, application, email

      Description automatically generated
2. In the ForEach control on the Activities tab, click the pencil to open the canvas specific to the ForEach activity. This is where we will write most of the logic.
3. Add a Lookup activity to the canvas of the IterateSQLTables ForEach activity and name it **GetChangeCount**.
   * 1. Under the Settings tab, create a new Dataset that points to the AdventureWorks SQL Dedicated Pool. Under the connection tab of the dataset, select the Dedicated SQL pool and leave the Table blank.

Graphical user interface, text, application, email

Description automatically generated

* + 1. Back on the GetChangeCount settings tab, select the following values:
       1. SQL pool: AdventureWorks
       2. Use query: query
       3. Query: select count(\*) as changecount from [@{item().TargetTableSchema}].[@{item().TargetTableName}]  
          **NOTE**: We are using this query to determine if there are any new records residing in the staging table of the Dedicated SQL Pool. If none exist, we will skip that table. This will be defined in the **If Condition** activity that we will build in the next steps.
       4. First row only: Checked  
          Graphical user interface, text, application, email

          Description automatically generated

1. Add an **If Condition** activity to the canvas to the right of **GetChangeCount** and name it **HasChangedRows**.
   1. Link the output of **GetChangeCount** activity to the input of **HasChangedRows.**
   2. On the Acitivites tab of the HasChangedRows activity, set the expression to  
      @greater(int(activity('GetChangeCount').output.firstRow.changecount),0)  
      **NOTE**: This will trigger a true or false based on the row count. If we have data in the staging table, it will trigger a true activity.
2. Click the pencil on the True Activity Listing of the Activities tab to activate a new canvas under the **HasChangedRows** activity. Up top, your breadcrumbs should look like the following.  
   
3. Next add a Stored Procedure Activity to the canvas and name it **EXEC spInsertUpdateDelete**.
   1. Under the Settings tab set the following values:
      1. Linked Service: This is tied to the Synapse Instance
      2. DB Name: AdventureWorks
      3. Stored procedure name: [SalesLT].[spInsertUpdateDelete]
      4. Edit: Checked
         1. Add 2 Stored Procedure Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| TableSchema | String | @{item().SourceTableSchema} |
| TableName | String | @{item().SourceTableName} |

Graphical user interface, text, application, email

Description automatically generated

1. Next we want to click back to the beginning of the breadcrumbs so we are at the root of the **Iterate-and-Exec-sp** pipeline.  
   Graphical user interface, text, application, chat or text message

   Description automatically generated
2. Now reopen the **GetTableList-Trigger-sp** pipeline and add an Execute Pipeline activity to the right of the **LookupTableList** activity and name it **Trigger sp**.
   1. On the Settings tab, set the following values
      1. Invoked Pipeline: Iterate-and-Exec-sp
      2. Wait on completion: Checked
      3. Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Value |
| tableList | array | @activity('LookupTableList').output.value |

Graphical user interface, text, application

Description automatically generated

1. Link the output of the **LookupTableList** activity to the input of the **Trigger sp** activity so the canvas looks like the following.  
   A picture containing diagram

   Description automatically generated
2. Publish your pipelines, datasets, etc.
3. Now we can test this pipeline. If the previous pipeline of **GetTableList-Trigger-Incremental-Copy** completed successfully, we should have records in the Staging.Address and Staging.Customer tables in the AdventureWorks SQL Dedicated Pool.   
   **NOTE:** If these records do not exist, nothing will happen when we trigger this pipeline.
4. Open the **GetTableList-Trigger-sp** and trigger the pipeline.
   1. When complete you should now have new rows in the [SalesLT].[Address] and [SalesLT].[Customer] tables of the AdventureWorks SQL Dedicated Pool.
   2. Also there should be no rows in the [Staging].[Address] and [Staging].[customer] tables of the AdventureWorks SQL Dedicated Pool as the last step of the stored procedure truncates these tables.