Lab: Import data with PolyBase & COPY using T-SQL and Copy Activity

Duration: 40 minutes

The main task for this exercise is as follows:

1. Import data with PolyBase and COPY using T-SQL
2. Petabyte-scale ingestion with Azure Synapse Pipelines

Task 1: Import data with PolyBase and COPY using T-SQL

Create staging tables

The **Sale** table has a columnstore index to optimize for read-heavy workloads. It is also used heavily for reporting and ad-hoc queries. To achieve the fastest loading speed and minimize the impact of heavy data inserts on the **Sale** table, it is decided to create a staging table for loads.

In this task, you will create a new staging table named **SaleHeap** in a new schema named **wwi_staging**. You will define it as a heap and use round-robin distribution. When WWI finalizes their data loading pipeline, they will load the data into **SaleHeap**, then insert from the heap table into **Sale**. Although this is a two-step process, the second step of inserting the rows to the production table does not incur data movement across the distributions.

1. In Synapse Analytics Studio, navigate to the **Develop** hub.
2. In the **+** menu, select **SQL script**.
3. In the toolbar menu, connect to the **SQLPool01** database.
4. In the query window, enter the following code to create the **wwi_staging** schema.

```
CREATE SCHEMA wwi_staging;
```

5. In the query window, replace the script with the following to create the heap table:

```
CREATE TABLE [wwi_staging].[SaleHeap]
(
    [TransactionId] [uniqueidentifier]  NOT NULL,
    [CustomerId] [int]  NOT NULL,
    [ProductId] [smallint]  NOT NULL,
    [Quantity] [smallint]  NOT NULL,
    [Price] [decimal](9,2)  NOT NULL,
    [TotalAmount] [decimal](9,2)  NOT NULL,
    [TransactionDate] [int]  NOT NULL,
    [ProfitAmount] [decimal](9,2)  NOT NULL,
    [Hour] [tinyint]  NOT NULL,
    [Minute] [tinyint]  NOT NULL,
    [StoreId] [smallint]  NOT NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    HEAP
)
GO
```

6. Run the SQL script to create the table.

7. In the query window, replace the script with the following to create the **Sale** table in the **wwi_staging** schema for load comparisons:

```
CREATE TABLE [wwi_staging].[Sale]
(
    [TransactionId] [uniqueidentifier]  NOT NULL,
    [CustomerId] [int]  NOT NULL,
    [ProductId] [smallint]  NOT NULL,
    [Quantity] [smallint]  NOT NULL,
    [Price] [decimal](9,2)  NOT NULL,
    [TotalAmount] [decimal](9,2)  NOT NULL,
    [TransactionDate] [int]  NOT NULL,
    [ProfitAmount] [decimal](9,2)  NOT NULL,
    [Hour] [tinyint]  NOT NULL,
    [Minute] [tinyint]  NOT NULL,
    [StoreId] [smallint]  NOT NULL
)
WITH
(
    DISTRIBUTION = HASH ( [CustomerId] ),
    CLUSTERED COLUMNSTORE INDEX,
    PARTITION
    (
        [TransactionDate] RANGE RIGHT FOR VALUES (20190101, 20190201, 20190301, 20190401,
20190501, 20190601, 20190701, 20190801, 20190901, 20191001, 20191101, 20191201)
    )
)
```

8. Run the script to create the table.

Configure and run PolyBase load operation

PolyBase requires the following elements:

- An external data source that points to the **abfss** path in ADLS Gen2 where the Parquet files are located
- An external file format for Parquet files
- An external table that defines the schema for the files, as well as the location, data source, and file format

1. In the query window, replace the script with the following code to create the external data source. Be sure to replace ***datalakexxxx*** with the your data lake storage account name:

```
CREATE EXTERNAL DATA SOURCE ABSS
WITH
( TYPE = HADOOP,
    LOCATION = 'abfss://wwi-02@datalakexxxxx.dfs.core.windows.net'
);
```

2. Run the script to create the external data source.

3. In the query window, replace the script with the following code to create the external file format and external data table. Notice that we defined **TransactionId** as an **nvarchar(36)** field instead of **uniqueidentifier**. This is because external tables do not currently support **uniqueidentifier** columns:

```
CREATE EXTERNAL FILE FORMAT [ParquetFormat]
WITH (
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
)
GO

CREATE SCHEMA [wwi_external];
GO

CREATE EXTERNAL TABLE [wwi_external].Sales
    (
        [TransactionId] [nvarchar](36)  NOT NULL,
        [CustomerId] [int]  NOT NULL,
        [ProductId] [smallint]  NOT NULL,
        [Quantity] [smallint]  NOT NULL,
        [Price] [decimal](9,2)  NOT NULL,
        [TotalAmount] [decimal](9,2)  NOT NULL,
        [TransactionDate] [int]  NOT NULL,
        [ProfitAmount] [decimal](9,2)  NOT NULL,
        [Hour] [tinyint]  NOT NULL,
        [Minute] [tinyint]  NOT NULL,
        [StoreId] [smallint]  NOT NULL
    )
WITH
```

```
    (
        LOCATION = '/sale-small/Year=2019',
        DATA_SOURCE = ABSS,
        FILE_FORMAT = [ParquetFormat]
    )
GO
```

Note: The */sale-small/Year=2019/* folder's Parquet files contain **4,124,857 rows**.

4. Run the script.

5. In the query window, replace the script with the following code to load the data into the **wwi_staging.SalesHeap** table:

```
INSERT INTO [wwi_staging].[SaleHeap]
SELECT * FROM [wwi_external].[Sales]
```

6. Run the script. It may take some time to complete.

7. In the query window, replace the script with the following to see how many rows were imported:

```
SELECT COUNT(1) FROM wwi_staging.SaleHeap
```

8. Run the script. You should see a result of 4124857.

Configure and run the COPY statement

Now let's see how to perform the same load operation with the COPY statement.

1. In the query window, replace the script with the following to truncate the heap table and load data using the COPY statement. As you did before, be sure to replace `datalakexxxxx` with your data lake name:

```
TRUNCATE TABLE wwi_staging.SaleHeap;
GO

COPY INTO wwi_staging.SaleHeap
FROM 'https://datalakexxxxx.dfs.core.windows.net/wwi-02/sale-small/Year=2019'
WITH (
    FILE_TYPE = 'PARQUET',
    COMPRESSION = 'SNAPPY'
)
GO
```

2. Run the script. Notice how little scripting is required to perform a similar load operation.

3. In the query window, replace the script with the following to see how many rows were imported:

```
SELECT COUNT(1) FROM wwi_staging.SaleHeap(nolock)
```

4. Run the script. Once again, 4124857 rows should have been imported. Note that both load operations copied the same amount of data in roughly the same amount of time.

Task 2: Petabyte-scale ingestion with Azure Synapse Pipelines

Configure workload management classification

When loading a large amount of data, it is best to run only one load job at a time for fastest performance. If this isn't possible, run a minimal number of loads concurrently. If you expect a large loading job, consider scaling up your dedicated SQL pool before the load.

Be sure that you allocate enough memory to the pipeline session. To do this, increase the resource class of a user which has permissions to rebuild the index on this table to the recommended minimum.

To run loads with appropriate compute resources, create loading users designated for running loads. Assign each loading user to a specific resource class or workload group. To run a load, sign in as one of the loading users, and then run the load. The load runs with the user's resource class.

1. In the query window, replace the script with the following to create a new login in **master** database.

   ```
   create login [import01user] with password = 'Pa@123456789';
   ```

2. In the query window, replace the script with the following to create a new user in **SQLPool01** database, assign db_owner role and create a new table.

   ```
   create user [import01user] for login [import01user];
   GO

   execute sp_addrolemember 'db_owner', 'import01user';
   GO

   Create Schema wwi_perf;
   GO

   CREATE TABLE [wwi_perf].[Sale_Heap]
   (
          [TransactionId] [uniqueidentifier]  NOT NULL,
          [CustomerId] [int]  NOT NULL,
          [ProductId] [smallint]  NOT NULL,
          [Quantity] [tinyint]  NOT NULL,
          [Price] [decimal](9,2)  NOT NULL,
          [TotalAmount] [decimal](9,2)  NOT NULL,
          [TransactionDateId] [int]  NOT NULL,
          [ProfitAmount] [decimal](9,2)  NOT NULL,
          [Hour] [tinyint]  NOT NULL,
          [Minute] [tinyint]  NOT NULL,
          [StoreId] [smallint]  NOT NULL
   )
   WITH
   ```

```
(
	DISTRIBUTION = ROUND_ROBIN,
	HEAP
)
```

3. In the SQL script query window you worked with in the previous exercise, replace the script with the following to create:

    - A workload group, **BigDataLoad**, that uses workload isolation by reserving a minimum of 50% resources with a cap of 100%
    - A new workload classifier, **HeavyLoader** that assigns the **import01user** user to the **BigDataLoad** workload group.

    At the end, we select from **sys.workload_management_workload_classifiers** to view all classifiers, including the one we just created:

```
-- Drop objects if they exist
IF EXISTS (SELECT * FROM sys.workload_management_workload_classifiers WHERE [name] =
'HeavyLoader')
BEGIN
    DROP WORKLOAD CLASSIFIER HeavyLoader
END;

IF EXISTS (SELECT * FROM sys.workload_management_workload_groups WHERE name =
'BigDataLoad')
BEGIN
    DROP WORKLOAD GROUP BigDataLoad
END;

--Create workload group
CREATE WORKLOAD GROUP BigDataLoad WITH
  (
      MIN_PERCENTAGE_RESOURCE = 50, -- integer value
      REQUEST_MIN_RESOURCE_GRANT_PERCENT = 25, --  (guaranteed min 4 concurrency)
      CAP_PERCENTAGE_RESOURCE = 100
  );

-- Create workload classifier
CREATE WORKLOAD Classifier HeavyLoader WITH
(
    Workload_Group ='BigDataLoad',
    MemberName='import01user',
    IMPORTANCE = HIGH
);

-- View classifiers
SELECT * FROM sys.workload_management_workload_classifiers
```

4. Run the script, and if necessary, switch the results to **Table** view. You should see the new **HeavyLoader** classifier in the query results:

5. Navigate to the **Manage** hub.

6. Create **Linked services,** Select Manage hub -> Linked service -> + New, then search Azure Synapse Analytics and select it and provide the following details:

    - Name **import01user**
    - From Azure Subscription: Checked
    - Azure Subscription: Select you subscription
    - Server name: synapsexx
    - Database Name: SQLPool01
    - Authentication type: SQL Authentication
    - Username: import01user
    - Password: enter your sqlpool password

    Test the connection and then create.

7. Notice that the username for the dedicated SQL pool connection is **import01user** user you added to the **HeavyLoader** classifier. We will use this linked service in our new pipeline to reserve resources for the data load activity.

Create pipeline with copy activity

1. Navigate to the **Integrate** hub.
2. In the **+** menu, select **Pipeline** to create a new pipeline.
3. In the **Properties** pane for the new pipeline, set the **Name** of the pipeline to `Copy December Sales`.
   Tip: After setting the name, hide the **Properties** pane.
4. Expand **Move & transform** within the Activities list, then drag the **Copy data** activity onto the pipeline canvas.
5. Select the **Copy data** activity on the canvas. Then, beneath the canvas, on the **General** tab, set the **Name** of the activity to `Copy Sales`.
6. Select the **Source** tab, then select **+ New** to create a new source dataset.
7. Select the **Azure Data Lake Storage Gen2** data store, then select **Continue**.
8. Choose the **Parquet** format, then select **Continue**.
9. In the **Set properties** pane:
   i. Set the name to `asal400_december_sales`.
   ii. Select the **datalake*xxxxxxx*** linked service (Your Data Lake Storage Linked Service).
   iii. Browse to the **wwi-02/campaign-analytics/sale-20161230-snappy.parquet** file
   iv. Select **From sample file** for schema import.
   v. Browse to **\dp-203\data-engineering-ilt-deployment\Allfiles\samplefiles\sale-small-20100102-snappy.parquet** in the **Select file** field.
   vi. Select **OK**.

   We downloaded a sample Parquet file that has the exact same schema, but is much smaller. This is because the file we are copying is too large to automatically infer the schema in the copy activity source settings.
10. Select the **Sink** tab, then select **+ New** to create a new sink dataset.
11. Select the **Azure Synapse Analytics** data store, then select **Continue**.
12. In the **Set properties** pane:
    i. Set the **Name** to `asal400_saleheap_asa`
    ii. Select the **import01user** linked service.

iii. Select the **wwi_perf.Sale_Heap** table

iv. Select **OK**.

13. In the **Sink** tab, select the **Copy command** copy method and enter the following code in the pre-copy script to clear the table before import:

    ```
    TRUNCATE TABLE wwi_perf.Sale_Heap
    ```

    The fastest and most scalable way to load data is through PolyBase or the COPY statement, and the COPY statement provides the most flexibility for high-throughput data ingestion into the SQL pool.

14. Select the **Mapping** tab and select **Import schemas** to create mappings for each source and destination field. Select **TransactionDate** in the source column to map it to the **TransactionDateId** destination column.

15. Select the **Settings** tab and set the **Data integration unit** to **8**. This is required due to the large size of the source Parquet file.

16. Select **Publish all**, then **Publish** to save your new resources.

17. Select **Add trigger**, then **Trigger now**. Then select **OK** in the **Pipeline run** pane to start the pipeline.

18. Navigate to the **Monitor** hub.

19. Select **Pipeline Runs**. You can see the status of your pipeline run here. Note that you may need to refresh the view. Once the pipeline run is complete, you can query the **wwi_perf.Sale_Heap** table to view the imported data.