Migrating a SQL Server to SQL DW

Lab04: Data loading best practices

Technologies showcased: SQL Data Warehouse, Azure Blob Storage

Author: Casey Karst (Cakarst@microsoft.com)

Program Manager II

Date: April 2017

Table of Contents

[Summary 3](#_Toc481749646)

[Pre-requisites 4](#_Toc481749647)

[Scenario 4](#_Toc481749648)

[Part 1 – Loading data into Azure SQL Data Warehouse 4](#_Toc481749649)

[Part 2 – Load Dimension tables 11](#_Toc481749650)

[Part 3 – Create Partitioned Fact Table 15](#_Toc481749651)

[Part 4 – Load data into partitioned staging tables from WASB 18](#_Toc481749652)

[Part 5 – Copy Data into Correctly formatted tables via CTAS 19](#_Toc481749653)

[Part 6 – Dynamic Management Views 20](#_Toc481749654)

[IMPORTANT: AVOID INCURRING EXTRA CHARGES - PAUSE YOUR SUBSCRIPTION RESOURCES 20](#_Toc481749655)

## Summary

This tutorial walks through loading best practices for a SQL Data Warehouse.

In this lab we will:

* Create necessary objects to connect to Windows Azure Blob Storage (WASB) container
* Load Dimension tables from a WASB container using best practices on distribution and indexes
* Load a Fact table data from WASB container into a partitioned table via staging tables
* Gain insight into the loading process via DMVs

The full set of scripts needed for this lab can be found on our VM under C:\Users\<UserName>\Desktop\LabContent\Day 2\Lab03

## Pre-requisites

* Azure Subscription with rights to use/deploy Azure services, and $200+ of Azure credit
* Software Requirement:
  + Visual Studio 2017 (SQL ops studio)
  + Portal query editor
  + SQL Server Data Tools 2016 (SSDT).
* SQL Data Warehouse
* Data in Windows Azure Storage Blob
* Access via portal to the storage blob created in Lab03

## Scenario

|  |  |  |
| --- | --- | --- |
| Part 1 – Loading data into Azure SQL Data Warehouse | | |
| **Scenario** | | |
| We have created our SQL Data Warehouse and now we want to load data into it. We can do this through the traditional ways of ETL and tooling such as SQL Server Integration Services or third-party tooling. However, today we are going to use Polybase. Your source data has been precreated and is in your Azure Blob Storage account. | | |
| **Commentary / Notes** | **Click Steps & ‘Bits’** | **Screenshots** |
| We need to open our data management toolset in order to prepare the data warehouse with a security, schema and data. SQL Server Data Tools (available within all versions of Visual Studio 2015) is the toolset to use within Visual Studio 2015 for Azure SQL Data Warehouse.  Once Visual Studio has opened we should set the pane view to support connectivity to Azure SQL Data Warehouse. | 1. Navigate to the Azure portal within the web browser which should be open from the last exercise. If not, open the browser and navigate to <https://portal.azure.com> 2. Open the Azure SQL Data Warehouse blade from the tile on the portal dashboard (you pinned it in the earlier exercise). |  |
|  | 1. Looking at the Overview blade you can see the Common Tasks as shown in the screen shot. |  |
|  | 1. Click the ‘Open in Visual Studio’ button.   *Note: Before opening Visual Studio click on Configure your firewall to make sure that your ClientIP has been added to the rules.* |  |
|  | 1. Click ‘Allow’ on the dialog box that appears. Visual Studio will now launch. |  |
|  | 1. Fill in the password specified when you created your data warehouse. 2. Click “Connect”. |  |
| Open SQL Server Object explorer window. Pin it. | 1. Expand the object tree within the SQL Server object explorer pane. |  |
| In the next set of steps we’re going to generate initial loading objects.  The first thing we are going to do is create the initial set of databse objects on your SQL DW that are required for loading.  These objects include:   * Database Master Key * Database Scoped Credential * External Data Source * External File Format   They are also reusable for future loads. | 1. Right click the database name and select ‘**New Query**’. A new query window will open. |  |
| Create Database Master Key: This object is used to encrypt the Database Scoped Credential's Secret and store it in the Data Warehouse | 1. Run the following Create Master Key script in your query window. Please choose your own password. | CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Password';  GO |
| Next, you will connect to the azure storage blob that you created in Lab:01.  When connecting to Azure Blob Storage, you need to supply the Azure Blob Storage Access key SQL DW so that it can access the data.  This is done via the Database Scoped Credential Secret. The identity is a required field, but it is not used for authentication. | 1. To retrieve your Key, go to the azure portal at <https://portal.azure.com> 2. Select the storage blob you created in Lab:01 |  |
|  | 1. In the Blob Overview menu under **Settings** select **Access Keys** under |  |
|  | 1. Under **Default keys** copy one of the Storage Access Keys |  |
| Next you’re going to create a Database Scoped Credential pasting in the key you copied as the secret.  Note: The Identity value needs to be filled in but is not used for loading | 1. Use the StorageAccountKey as the secret into the following script in your query window 2. Run the following query | Create Database Scoped Credential Mastdata  with Identity = 'user',  Secret = '<StorageAccountKey>' |
| In the next step we’re going to create our External Data Source. For this step you will need your Windows Azure Storage Blob (WASB) Uri | 1. To find your WASBI Uri, go to the azure portal at <https://portal.azure.com> 2. Select the storage blob you created in Lab:01 3. The blob will open the container you created in Lab:01 4. Click on **Properties** in the top menu |  |
|  | 1. The properties pane will appear on the right 2. Select the container Url and paste it into your query window. |  |
|  | 1. Next you will need to restructure the Url as follows: | Copied:  https://<StrorageAcountName>.blob.core.windows.net/<ContainerName>  Updated:  wasbs://<ContainerName>@<StrorageAcountName>.data.blob.core.windows.net/ |
| In the next step you will create your external data source using the WASB Uri and the Database scoped credential you just created | 1. Use the following query with the new WASB Uri you just created | CREATE EXTERNAL DATA SOURCE MastData\_stor  WITH  (  TYPE = HADOOP,  LOCATION = N'< wasbs://<ContainerName>@<StrorageAcountName>.data.blob.core.windows.net/>',  CREDENTIAL = Mastdata  ) |
| Next, we’re going to define the external file format of the data stored in your WASB location. For this exercise we will be using a pipe delimited text file. | 1. Run the following query in your query window | CREATE EXTERNAL FILE FORMAT pipe  WITH (FORMAT\_TYPE = DELIMITEDTEXT,  FORMAT\_OPTIONS(  FIELD\_TERMINATOR = '|',  STRING\_DELIMITER = '',  DATE\_FORMAT = '',  USE\_TYPE\_DEFAULT = False)  ) |
| End of part 1 | | |

|  |  |  |
| --- | --- | --- |
| Part 2 – Load Dimension tables | | |
| **Scenario** | | |
| Now, we have created our external data source we can query and load the data we have in the Azure Blob Store.  In the following lab we will load dimension tables into our SQL DW. Dimension tables are often a good first step because they are relatively small and this will allow you to gain an understanding of how to load data into SQL DW from WASB. | | |
| To load data from WASB, you first create an external table over the external data and then run Create Table as Select (CTAS) command. In the following steps we will do the following:   * Create and external table * Write a CTAS statement * Update the Statics on the table we created   The end result is dimension tables with data. | 1. Using the following script to create an external table called Aircraft\_IMPORT   Note:   * **data\_source** - References the External Data Source that you want to read from * **File\_format** - References the File Format that the data is in * **location** - Specifies the directory location that you want to read from. PolyBase traverses all childern directories and files from a stated filepath. | CREATE EXTERNAL TABLE Aircraft\_IMPORT  ([id] [int] NULL,  [TailNum] [varchar](15) NULL,  [Type] [varchar](50) NULL,  [Manufacturer] [varchar](50) NULL,  [IssueDate] [varchar](15) NULL,  [Model] [varchar](20) NULL,  [Status] [char](5) NULL,  [AircraftType] [varchar](30) NULL,  [EngineType] [varchar](20) NULL,  [Year] [smallint] NULL)  WITH  (  DATA\_SOURCE = MastData\_Stor,  FILE\_FORMAT = pipe,  LOCATION = 'aircraft'  ) |
| The external table now acts as a pointer to data outside of the Data Warehouse.  Next were going to create the table and load data using Create Table as Select (CTAS) | 1. Use the following CTAS script to create the table and load data   Note:   1. Make sure that you select \* From Aircraft\_IMPORT you just created. | CREATE TABLE Dim\_Aircraft  WITH  (  DISTRIBUTION = ROUND\_ROBIN  , CLUSTERED INDEX (id)  )  AS SELECT \* FROM Aircraft\_IMPORT |
| Next we’re going to update the statistics on table you just created.  Statistics need to be created or updated any time that data is loaded into SQL DW. Statistics should be created on all columns that are used in joins and are frequently used in predicates | 1. Run the following script to update Statstics 2. Auto update statistics can take care of automatically updating single column stats, but in this case it is multi-column stats | CREATE STATISTICS Aircraft\_Stat  ON  Dim\_Aircraft (id, type, manufacturer) |
| Final were going to load the remainder of our dimension tables | 1. To load the remainder of the dimension tables. Run the **1 - Create Dimensions.dsql** script that can be found in the LabContent folder on your desktop under Day 2\Lab03 | Create External table Airport\_export  ([id] [int] NULL,  [Code] [nvarchar](10) NULL,  [Name] [varchar](50) NULL,  [City] [varchar](50) NULL,  [State] [varchar](15) NULL,  [Country] [varchar](50) NULL,  [Latitude] [decimal](9, 6) NULL,  [Longitude] [decimal](9, 6) NULL,  [geo] varchar(1000) NULL)  With  (  data\_source = MastData\_Stor,  File\_format = pipe,  location = 'Airport'  ) |
| CREATE TABLE Dim\_Airport  WITH  (  Distribution = ROUND\_ROBIN, Clustered INDEX (id)  )  AS  SELECT \* FROM Airport\_export  CREATE Statistics Airport\_Stat on Dim\_Airport(id, Name, Country)  Create External Table Carrier\_export  ([ID] [smallint] NOT NULL,  [Code] [nvarchar](10) NULL,  [Description] [varchar](100) NULL)  WITH  (  data\_source = MastData\_Stor,  File\_format = pipe,  location = 'Carrier'  )  CREATE TABLE Dim\_Carrier  WITH  (  Distribution = ROUND\_ROBIN, Clustered INDEX (id)  )  AS  SELECT \* FROM Carrier\_export  CREATE Statistics Carrier\_Stat on Dim\_Carrier(id, code)  Create External Table [Date\_Export](  [ID] [int] NOT NULL,  [Date] [date] NOT NULL,  [DayOfMonth] [tinyint] NOT NULL,  [DayofWeek] [tinyint] NOT NULL,  [DayName] [varchar](9) NOT NULL,  [MonthOfYear] [tinyint] NOT NULL,  [MonthName] [varchar](9) NOT NULL,  [Year] [smallint] NOT NULL,  [IsWeekend] [bit] NOT NULL,  [SeasonId] [tinyint] NOT NULL,  [Season] [varchar](10) NOT NULL)  With  (  data\_source = MastData\_Stor,  File\_format = pipe,  location = 'date'  ) |
|  |  | CREATE TABLE Dim\_Date  WITH  (  Distribution = ROUND\_ROBIN, Clustered INDEX (id)  )  AS  SELECT \* FROM [Date\_Export]  CREATE Statistics Date\_Stat on Dim\_Date(id, MonthName, Year)  Create external table Time\_Export(  [ID] [smallint] NOT NULL,  [Time] [time](0) NOT NULL,  [HourOfDay] [tinyint] NOT NULL,  [PartofDayId] [tinyint] NOT NULL,  [PartOfDay] [varchar](10) NOT NULL)  With  (  data\_source = MastData\_Stor,  File\_format = pipe,  location = 'Time'  )  CREATE TABLE Dim\_Time  WITH  (  Distribution = ROUND\_ROBIN, Clustered INDEX (id)  )  AS  SELECT \* FROM Time\_Export  CREATE Statistics Time\_Stat on Dim\_Time(id) |
| End of Part 2 | | |

|  |  |  |
| --- | --- | --- |
| Part 3 – Create Partitioned Fact Table | | |
| **Scenario** | | |
| To effectively leverage a partition swap load, a table has to exist with an exisiting partition scheme. To do this you must create an empty table with a partitioning scheme. | | |
| In order to use partition switching functionality, you need to first add a table to partition switch into. | 1. To create an empty table partioned by DateID. Run the **2 - Create Fact Table.dsql** script that can be found in the LabContent folder on your desktop under Day 2\Lab03 |  |
| Next we’re going to load our fact tables from WASB.  Because the directory structure of the table in azure is tablename/year/month we can take advantage of this and break the load into several segments.  Our partition scheme is based on year, so we load the data in yearly batches. | 1. To load the staging tables from WASB into SQDL DW. Run the following script that is part of **3 - InitialFactLoad.dsql** script that can be found in the LabContent folder on your desktop under Day 2\Lab03 |  |
| Next we’ll start the import loop that will load the table.  Each iteration of the loop will create an external table, load the data from that external table into an internal table, and then drop the external table.  The table names have a suffix which identifies the starting date. | Note:   * We are using Round\_Robin distribution and a Heap because we want to ensure that the load occurs as quickly as possible. Remember ELT. * Use CTAS external table and create a local table. |  |
| End of Part 3 | | |

|  |  |  |
| --- | --- | --- |
| Part 4 – Load data into partitioned staging tables from WASB | | |
| **Scenario** | | |
| In the next set of steps we are going to take the staging tables we created in part 3 and prep the data for a partition switch. | | |
| In order to be able to switch a partition from one table into another the following has to be true:   1. The distribution has to be the same 2. The index has to be the same 3. The partition boundaries have to be the same | 1. To complete the staging table prep. Run the following script that is part of **4 -PartitionStagingTables.dsql** script that can be found in the LabContent folder on your desktop under Day 2\Lab03 |  |
| End of Part 4 | | |

|  |  |  |
| --- | --- | --- |
| Part 5 – Copy Data into Correctly formatted tables via CTAS | | |
| **Scenario** | | |
| Now that we have a set of partitioned tables and an empty fact table, we can start doing partition switches into the table. | | |
| The next script that you will run loops through the partitioned tables and dynamically switches the partitions. Because this operation is on the metadata, there is relatively little downtime for the amount of data "loaded" into the production fact table. | 1. To switch the partitions on your empty fact table. Run the following script that is part of **5 - LoadWithPartitionSwitch.dsql** script that can be found in the LabContent folder on your desktop under Day 2\Lab03 |  |
| End of Part 5 | | |

|  |  |  |
| --- | --- | --- |
| Part 6 – Dynamic Management Views | | |
| **Scenario** | | |
| Next we’ll query the Data warhouse to see how much data has been loaded. | | |
| You may note that the row count is less than optimal for CCI. Loading any more data would have been prohibitively slow for the lab.  The following query allows you to see the row counts per partition in both the final fact table and the staging table. | 1. To query the row count. Run the following script that is part of **6 - DMVs.dsql** script that can be found in the LabContent folder on your desktop under Day 2\Lab03 |  |
| End of Part 6 | | |

**IMPORTANT: AVOID INCURRING EXTRA CHARGES BY PAUSING YOUR SUBSCRIPTION RESOURCES**