APS to Azure SQL Data Warehouse Migration



Schema and Data Migration with PolyBase

Prepared for

1-Oct-20

Version 1.0 Draft

Contributors

**Arshad Ali,** Solution Architect

**Kalyan Yella, Solution Architect**

**Vishal Singh**, Consultant

**Basawashree Vasmate**, Consultant

Table of Contents

[1 Introduction 2](#_Toc486421940)

[1.1 Feature Highlights 4](#_Toc486421941)

[1.2 Sample script output – formatted for readability purpose 5](#_Toc486421942)

[2 Script Generation Automation Framework 11](#_Toc486421943)

[2.1 Exporting Data from APS to Blob Storage 11](#_Toc486421944)

[2.2 Importing Data from Blob Storage to SQL DW 14](#_Toc486421945)

[2.3 Generating scripts for other objects 16](#_Toc486421946)

[3 References 17](#_Toc486421947)

[4 Appendix 18](#_Toc486421948)

[4.1 Configure PolyBase Connectivity to External Data 18](#_Toc486421949)

1. Introduction

This document talks about *Script Generation Automation Framework* which can help you dynamically generate scripts for exporting table schema as well as data from APS to blob storage and then to create corresponding tables and import data back in SQL DW from the blob storage. The *Script Generation Automation Framework* also generates scripts for other types of objects, like views, stored procedures etc. The generated scripts then can be executed sequentially or in parallel by running them in multiple query windows.

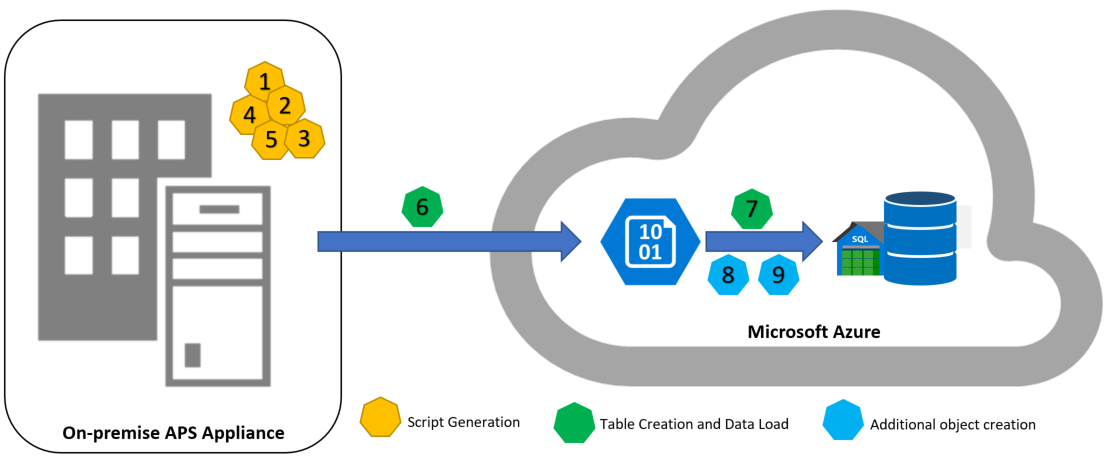


Figure - Schema and Data Migration - Workflow

1. Generate scripts (external table) for all tables in required source databases for the APS appliance.
2. Generate scripts (external table) for all tables in SQL DW corresponding to the tables from source.
3. Generate scripts (internal table) for all tables from required source databases for SQL DW, along with distribution, cluster indexing and partitioning information as they are defined in APS.
4. Optionally, generate scripts for additional supporting objects for tables in APS, like additional non-clustered indexes, statistics, default constraints etc. if applicable.
5. Generate scripts for modules (like views, stored procedures, functions etc.).
6. You can then execute scripts, created in step 1, on APS appliance to create external tables in APS databases and exporting data out from appliance to configured blob storage (refer appendix section below to learn how about configuring blob storage in APS to use with external tables).
7. Next, you can execute scripts, created in step 2, on SQL DW to created external tables in SQL DW and execute scripts, created in step 3, to create internal tables and load data into it (using CTAS from external tables). It also takes care of defining distribution, partitioning, indexes by deriving same structure from the APS databases.
8. Then, you can execute scripts, created in step 4, on SQL DW to create like additional non-clustered indexes, statistics, default constraints etc. if applicable in SQL DW.
9. Finally, you need to change (references of the table or view names in the code) and execute scripts, created in step 5, on SQL DW to create modules (like views, stored procedures, functions etc.) in SQL DW.

As cross database joins are not supported in SQL Data Warehouse, databases from source are consolidated in to a single destination SQL Data Warehouse and separated using schema. For example, for a source database STG and schema DBO the corresponding schema in SQL Data Warehouse will be STG\_DBO. Likewise, for a source database DM1 and schema FINANCE the corresponding schema in SQL Data Warehouse will be DM1\_FINANCE. Of course, this requires to changes references of the objects in views or modules.

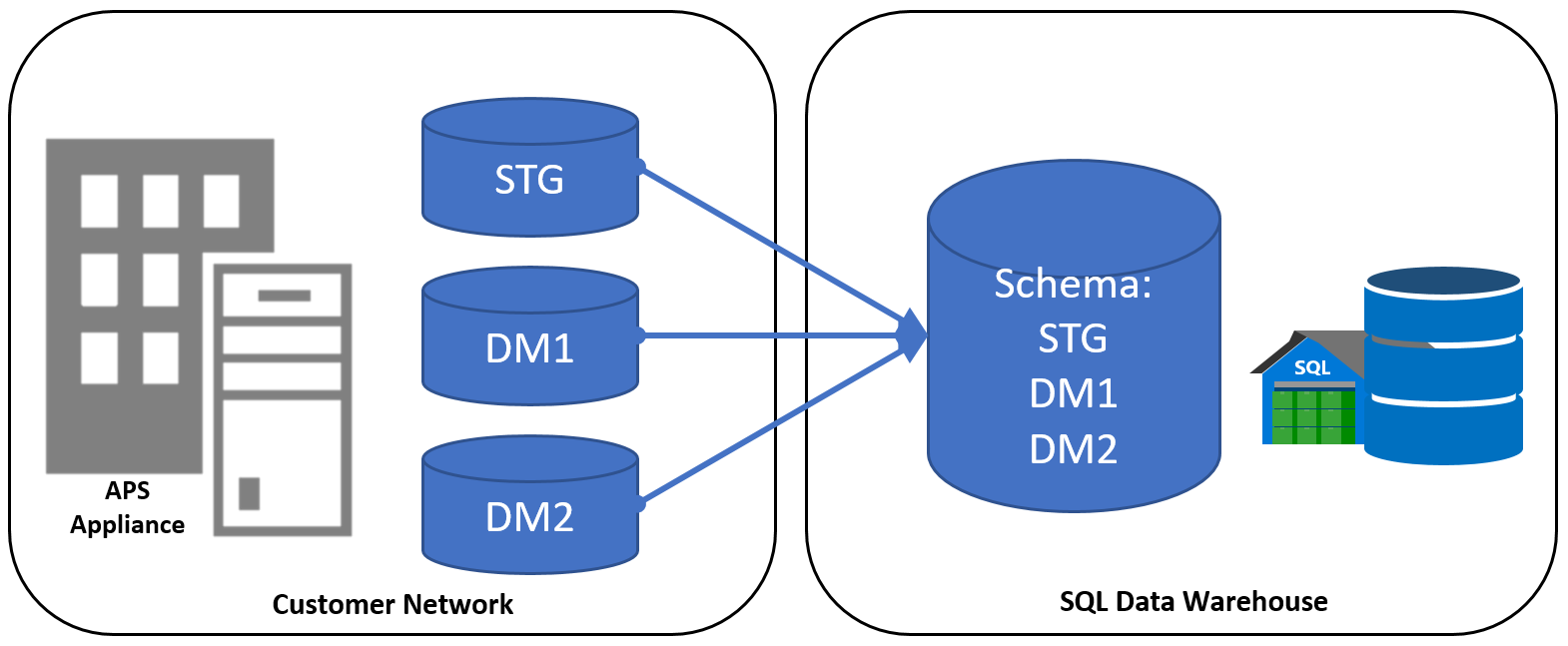


Figure - Schema Migration

For data migration, external tables are created in APS to export the data out to blob storage and then external tables are created in SQL Data Warehouse to reference the data from blob storage. Finally, internal tables are created and populated with data from blob storage via external tables in SQL Data Warehouse.

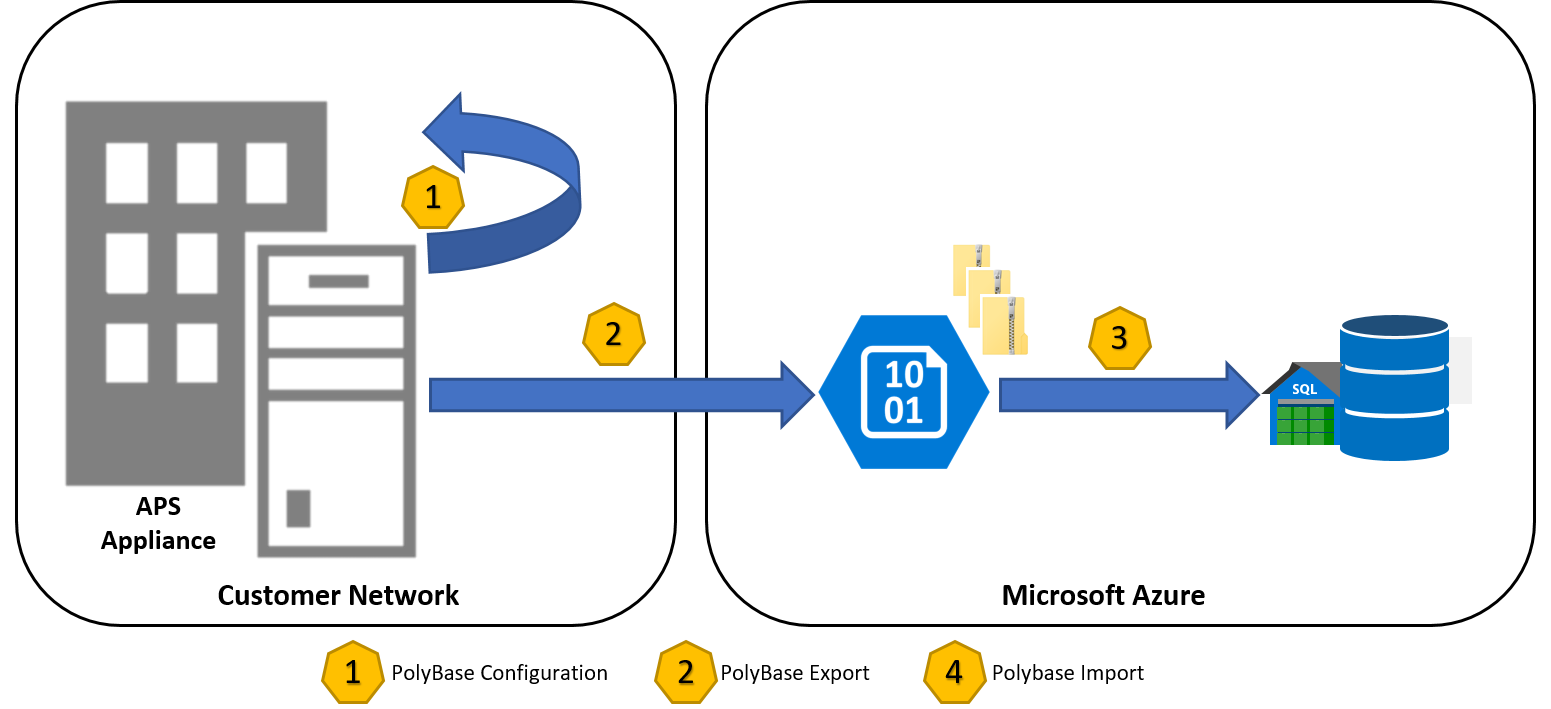


Figure - Data Migration with PolyBase

* 1. Feature Highlights
* The framework lets you specify multiple databases to consider for migration. It dynamically scales to all the those specified databases and consider them iteratively in one go.
* For internal tables in SQL DW, the framework derives partitioning strategy from APS for each of the tables and apply the same automatically for SQL DW internal tables.
* For internal tables in SQL DW, the framework derives indexing strategy from APS for each of the tables and apply the same automatically for SQL DW internal tables.
  + For clustered columnstore index when data load completes it creates clustered columnstore index on the SQL DW internal tables.
  + For clustered rowstore index, if one exists on APS table, it applies the same to internal tables in SQL DW, by defining all columns in the same sequence and in same ascending\descending order.
  + If there are any non-clustered index (or indexes) on APS table, that also gets created for respective internal table in SQL DW automatically.
* For internal tables in SQL DW, the framework derives statistics strategy from APS for each of the tables and apply the same automatically for respective table in SQL DW.
  1. Sample script output – formatted for readability purpose
     1. Create Data Source and File Format

Based on configuration parameter set, the framework generates data source and file format (creating master key and credential is not shown here):

|  |
| --- |
| IF EXISTS (SELECT \* FROM sys.external\_file\_formats WHERE name = 'ff\_textdelimited')  DROP EXTERNAL FILE FORMAT ff\_textdelimited;  CREATE EXTERNAL FILE FORMAT ff\_textdelimited  WITH (  FORMAT\_TYPE = DELIMITEDTEXT,  FORMAT\_OPTIONS (  FIELD\_TERMINATOR = '^|^',  DATE\_FORMAT = 'MM/dd/yyyy'),  DATA\_COMPRESSION = 'org.apache.hadoop.io.compress.GzipCodec'  );  IF EXISTS (SELECT \* FROM sys.external\_data\_sources WHERE name = 'ds\_blobstorage')  DROP EXTERNAL DATA SOURCE ds\_blobstorage;  CREATE EXTERNAL DATA SOURCE ds\_blobstorage  WITH (  TYPE = HADOOP,  LOCATION = 'wasbs://container@blobaccount.blob.core.windows.net/'  ); |

* + 1. Create Schema in APS for external tables

Based on configuration parameter set, framework creates a schema in APS so that all the objects are segregated into for manageability purpose:

|  |
| --- |
| USE AdventureWorksPDW2012;  IF EXISTS(SELECT \* FROM sys.schemas WHERE name = 'EXTSQLDW')  DROP SCHEMA [EXTSQLDW];  EXEC('CREATE SCHEMA [EXTSQLDW] AUTHORIZATION dbo;') |

* + 1. Create External Table - APS

This is just one sample script to create an external table in APS database. Based on configurable parameters set, you can include or exclude databases and tables, and framework takes into consideration of only identified databases\tables for migration. After execution of this output script, you will have data for this table exported to specified blob storage under folder specified.

|  |
| --- |
| IF EXISTS(SELECT \* FROM sys.external\_tables WHERE name = 'dbo.FactInternetSales')  DROP TABLE [EXTSQLDW].[dbo.FactInternetSales];  CREATE EXTERNAL TABLE [EXTSQLDW].[dbo.FactInternetSales]  WITH (  LOCATION = '/AdventureWorksPDW2012/dbo/FactInternetSales/',  DATA\_SOURCE = ds\_blobstorage,  FILE\_FORMAT = ff\_textdelimited,  REJECT\_TYPE = VALUE,  REJECT\_VALUE = 0  )  AS SELECT \* FROM AdventureWorksPDW2012.[dbo].[FactInternetSales]; |

* + 1. Create External Table – SQL DW

This is just one sample script to create an external table in SQL DW database. As you can notice, the framework derives structure of the table from table structure in APS database. Once this output script is executed on SQL DW database, it will simply point to the data in blob storage (no data is imported yet, data import happens with the next set of scripts).

|  |
| --- |
| IF NOT EXISTS(SELECT \* FROM AdventureWorksPDW2012.sys.schemas  WHERE name = 'AdventureWorksPDW2012\_dbo')  EXEC('CREATE SCHEMA [AdventureWorksPDW2012\_dbo] AUTHORIZATION dbo;');  IF EXISTS(SELECT \* FROM sys.external\_tables WHERE schema\_id = SCHEMA\_ID('AdventureWorksPDW2012\_dbo') AND name = 'EXT\_FactInternetSales')  DROP TABLE [AdventureWorksPDW2012\_dbo].[EXT\_FactInternetSales];  CREATE EXTERNAL TABLE [AdventureWorksPDW2012\_dbo].[EXT\_FactInternetSales]  (  [ProductKey] int NOT NULL,  [OrderDateKey] int NOT NULL,  [DueDateKey] int NOT NULL,  [ShipDateKey] int NOT NULL,  [CustomerKey] int NOT NULL,  [PromotionKey] int NOT NULL,  [CurrencyKey] int NOT NULL,  [SalesTerritoryKey] int NOT NULL,  [SalesOrderNumber] nvarchar(20) COLLATE Latin1\_General\_100\_CI\_AS\_KS\_WS NOT NULL,  [SalesOrderLineNumber] tinyint NOT NULL,  [RevisionNumber] tinyint NOT NULL,  [OrderQuantity] smallint NOT NULL,  [UnitPrice] money NOT NULL,  [ExtendedAmount] money NOT NULL,  [UnitPriceDiscountPct] float NOT NULL,  [DiscountAmount] float NOT NULL,  [ProductStandardCost] money NOT NULL,  [TotalProductCost] money NOT NULL,  [SalesAmount] money NOT NULL,  [TaxAmt] money NOT NULL,  [Freight] money NOT NULL,  [CarrierTrackingNumber] nvarchar(25) COLLATE Latin1\_General\_100\_CI\_AS\_KS\_WS NULL,  [CustomerPONumber] nvarchar(25) COLLATE Latin1\_General\_100\_CI\_AS\_KS\_WS NULL  )  WITH (  LOCATION = '/AdventureWorksPDW2012/dbo/FactInternetSales/',  DATA\_SOURCE = ds\_blobstorage,  FILE\_FORMAT = ff\_textdelimited,  REJECT\_TYPE = VALUE,  REJECT\_VALUE = 0  ); |

* + 1. Create Internal Table – ROUND\_ROBIN

This is just one sample script to create an internal table in APS database. As you can notice, the framework derives structure of the table from table structure in APS database as well as it derives distribution type, partitioning, index structure as well.

|  |
| --- |
| CREATE TABLE [AdventureWorksPDW2012\_dbo].[FactInternetSalesRR]  WITH (DISTRIBUTION = ROUND\_ROBIN,  PARTITION ([OrderDateKey] RANGE RIGHT FOR VALUES ([20000101],  [20010101], [20020101], [20030101], [20040101], [20050101],  [20060101], [20070101], [20080101], [20090101], [20100101],  [20110101], [20120101], [20130101], [20140101], [20150101],  [20160101], [20170101], [20180101], [20190101], [20200101],  [20210101], [20220101], [20230101], [20240101], [20250101],  [20260101], [20270101], [20280101], [20290101]))  )  AS  SELECT \* FROM [AdventureWorksPDW2012\_dbo].[EXT\_FactInternetSalesRR];  CREATE CLUSTERED COLUMNSTORE INDEX [cci\_AdventureWorksPDW2012\_dbo\_FactInternetSalesRR]  ON [AdventureWorksPDW2012\_dbo].[FactInternetSalesRR]; |

* + 1. Create Internal Table – REPLICATED

As REPLICATED tables are yet not supported in SQL DW, the framework uses ROUND\_ROBIN distribution for REPLICATED tables and put a comment inline so that it can be identified and changed quickly once REPLICATED table support is available in SQL DW.

|  |
| --- |
| CREATE TABLE [AdventureWorksPDW2012\_dbo].[FactInternetSalesR]  WITH (DISTRIBUTION = ROUND\_ROBIN /\*REPLICATE CHANGED TO ROUND\_ROBIN\*/,  PARTITION ([OrderDateKey] RANGE RIGHT FOR VALUES ([20000101], [20010101],  [20020101], [20030101], [20040101], [20050101], [20060101], [20070101],  [20080101], [20090101], [20100101], [20110101], [20120101], [20130101],  [20140101], [20150101], [20160101], [20170101], [20180101], [20190101],  [20200101], [20210101], [20220101], [20230101], [20240101], [20250101],  [20260101], [20270101], [20280101], [20290101])  ) )  AS  SELECT \* FROM [AdventureWorksPDW2012\_dbo].[EXT\_FactInternetSalesR];  CREATE CLUSTERED COLUMNSTORE INDEX [cci\_AdventureWorksPDW2012\_dbo\_FactInternetSalesR]  ON [AdventureWorksPDW2012\_dbo].[FactInternetSalesR]; |

* + 1. Create Internal Table – HASH

The framework derives distribution and hash key information from APS databases and creates internal tables in SQL DW with the same structure.

|  |
| --- |
| CREATE TABLE [AdventureWorksPDW2012\_dbo].[FactInternetSales]  WITH (DISTRIBUTION = HASH([OrderDateKey]), PARTITION ([OrderDateKey]  RANGE RIGHT FOR VALUES ([20000101], [20010101], [20020101],  [20030101], [20040101], [20050101], [20060101], [20070101],  [20080101], [20090101], [20100101], [20110101], [20120101],  [20130101], [20140101], [20150101], [20160101], [20170101],  [20180101], [20190101], [20200101], [20210101], [20220101],  [20230101], [20240101], [20250101], [20260101], [20270101],  [20280101], [20290101]))  )  AS  SELECT \* FROM [AdventureWorksPDW2012\_dbo].[EXT\_FactInternetSales];  CREATE CLUSTERED COLUMNSTORE INDEX [cci\_AdventureWorksPDW2012\_dbo\_FactInternetSales]  ON [AdventureWorksPDW2012\_dbo].[FactInternetSales]; |

* + 1. Create Internal Table – CLUSTERED

Often tables in APS have clustered columnstore index but few small tables might have clustered rowstore index, again the framework derive this information from source and creates table accordingly in SQL DW.

|  |
| --- |
| CREATE TABLE [AdventureWorksPDW2012\_dbo].[DimSalesReason]  WITH (DISTRIBUTION = ROUND\_ROBIN /\*REPLICATE CHANGED TO ROUND\_ROBIN\*/ )  AS  SELECT \* FROM [AdventureWorksPDW2012\_dbo].[EXT\_DimSalesReason];  CREATE CLUSTERED INDEX [ci\_AdventureWorksPDW2012\_dbo\_DimSalesReason]  ON [AdventureWorksPDW2012\_dbo].[DimSalesReason] ([SalesReasonKey] ASC) |

* + 1. Create Non-Clustered Index

Likewise, if an APS table has non-clustered indexes, the framework derives that information and creates those non-clustered indexes on corresponding SQL DW table as well.

|  |
| --- |
| TOBEUPDATED |

* + 1. Create default constraints

If an APS table has columns with default constraints, the framework derives that information and creates those default constraints once SQL DW table has been loaded with data, index and statistics have been created.

|  |
| --- |
| TOBEUPDATED |

* + 1. Create Statistics

By default, clustered columnstore index creates statistic on the table though if there are additional user created statistics. This framework identifies these additional statistics and create them once data load SQL DW table has been loaded with data and index have been created.

|  |
| --- |
| CREATE STATISTICS [OrderDatekey] ON [AdventureWorksPDW2012\_dbo].[FactInternetSales] ([OrderDateKey]);  CREATE STATISTICS [stat\_FactInternetSalesReason\_SalesOrderLineNumber] ON [AdventureWorksPDW2012\_dbo].[FactInternetSalesReason] ([SalesOrderLineNumber]);  CREATE STATISTICS [stat\_FactInternetSalesReason\_SalesOrderNumber] ON [AdventureWorksPDW2012\_dbo].[FactInternetSalesReason] ([SalesOrderNumber]);  CREATE STATISTICS [stat\_FactInternetSalesReason\_SalesReasonKey] ON [AdventureWorksPDW2012\_dbo].[FactInternetSalesReason] ([SalesReasonKey]); |

* + 1. Create Modules

Currently, though, the framework can export scripts for all the modules (views, stored procedures, functions) from APS, it cannot be directly executed on SQL DW as is. It needs to be manually edited to change references of the objects in the code, for example from 3-part naming to 2-part naming convention and then only it can be executed, or modules can be created on SQL DW.

|  |
| --- |
| CREATE PROC [dbo].[ETL\_LogEvent] @SPName [VARCHAR](100),@StepName [VARCHAR](100) AS  BEGIN  DECLARE @ID INT,  @EventDateTime Datetime  SET @ID=(SELECT ISNULL(MAX(ID),0) FROM [dbo].[ETL\_log])+1  SET @EventDateTime=GETDATE()  INSERT INTO [APS\_ETL\_Framework].[dbo].[ETL\_log]([ID],[SPName],[StepName],[EventDateTime])  VALUES(@ID,@SPName,@StepName,@EventDateTime)  END;  CREATE PROC [dbo].[ExecutionLogStart] @ExecutionID [VARCHAR](50),@SP [VARCHAR](100),@Section [VARCHAR](100),@Step [CHAR](5),@Message [VARCHAR](500),@Status [VARCHAR](16),@CreatedBy [INT] AS  BEGIN  DECLARE @CreatedOn DATETIME,  @Execution VARCHAR(50)  SET @Execution=@ExecutionID+@Step  SET @CreatedOn=GETDATE()  INSERT INTO [dbo].[Log\_SPExecution]  ([LogID],  [ExecutionID],  [StoredProc],  [Section],  [StartTime],  [Message],  [Status],  [CreatedOn],  [CreatedBy])  VALUES (@Execution,  @ExecutionID,  @SP,  @Section,  @CreatedOn,  @Message,  @Status,  @CreatedOn,  @CreatedBy  )  END |

1. Script Generation Automation Framework
   1. Exporting Data from APS to Blob Storage

Script file available in attachment can be used for automatically generating scripts for creating external tables in APS appliance and exporting data to Azure Blob Storage. **ExportToBlob-Part1.dsql** generates script for external tables but before execution you can specify a schema under which all these external tables will be created.

|  |
| --- |
| DECLARE @SchemaForExternalTable VARCHAR(255) = 'EXTSQLDW’ |

This allows you to specify databases to consider during migration. You can specify one or more databases to consider in single execution. For each database you want to include, you need to one INSERT statement for each database as shown below:

|  |
| --- |
| --step 2: define databases that you want to include  INSERT INTO DatabasesToInclude VALUES ('AdventureWorksPDW2012') |

After execution of the above script, output should look like this:

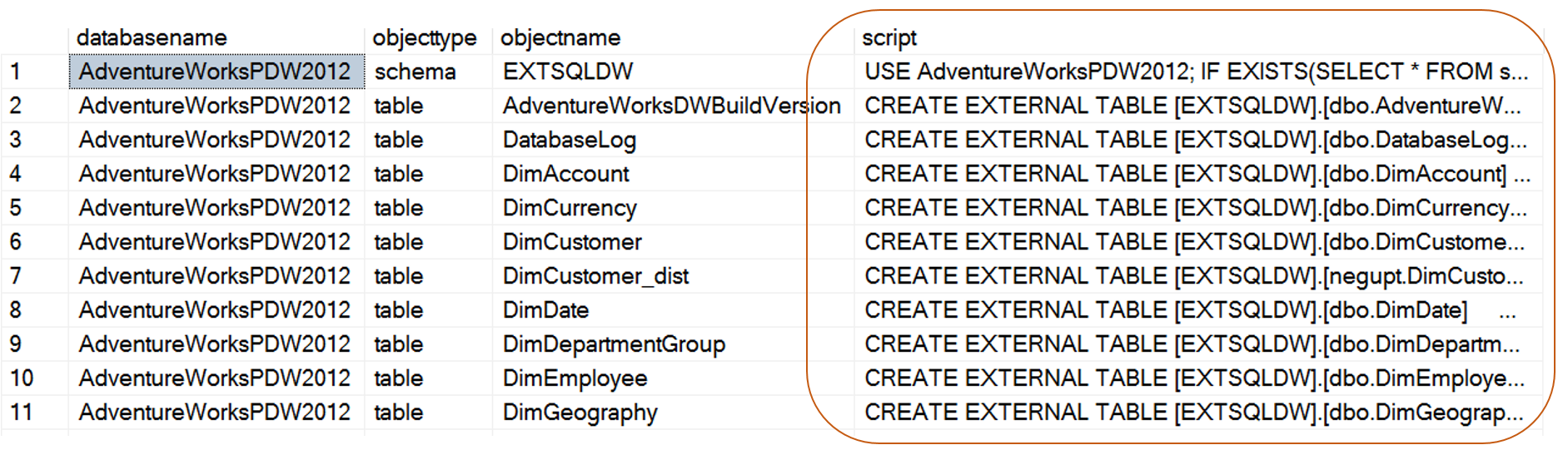


Figure - Export - Dynamic Script Generation

Next you need to execute **ExportToBlob-Part2.dsql** but before that you need to again specify few important configuration parameters, for example,

* @AzureStorageAccount - You need to specify the blob storage location where data needs to be exported. This storage account must have been setup in core-site.xml file and APS must have been restarted after that change (refer appendix section below to learn how about configuring blob storage in APS to use with external tables).
* @FieldDelimiter – You can specify a single character delimiter to have a compact data file but in case if you suspect a collision between one specific character delimiter with the data in tables, you can make it multi-characters.

|  |
| --- |
| --step 1: define all parameters  DECLARE @FormatType VARCHAR(100) = 'DELIMITEDTEXT'  DECLARE @FieldDelimiter VARCHAR(10) = '^|^'  DECLARE @DateFormat VARCHAR(12) = 'MM/dd/yyyy'  DECLARE @DataCompression VARCHAR(100) = 'org.apache.hadoop.io.compress.GzipCodec'  DECLARE @AzureStorageAccount VARCHAR(1000) = 'wasbs://<containername>@<accountname>.blob.core.windows.net/' |

Also, before execution of the **ExportToBlob-Part2.dsql** script you need to copy script output (after executing **ExportToBlob-Part1.dsql** script and as shown in Figure 2 above) to the end of the **ExportToBlob-Part2.dsql** file.

This copied script has two types of script for each of the databases and you need to execute it in this sequence (though its generated already in proper order),

* Execute first statement to switch database context and drop migration schema if already existing and create a new one.
* Finally, run all the scripts for creating external tables and exporting data to blob storage for that database

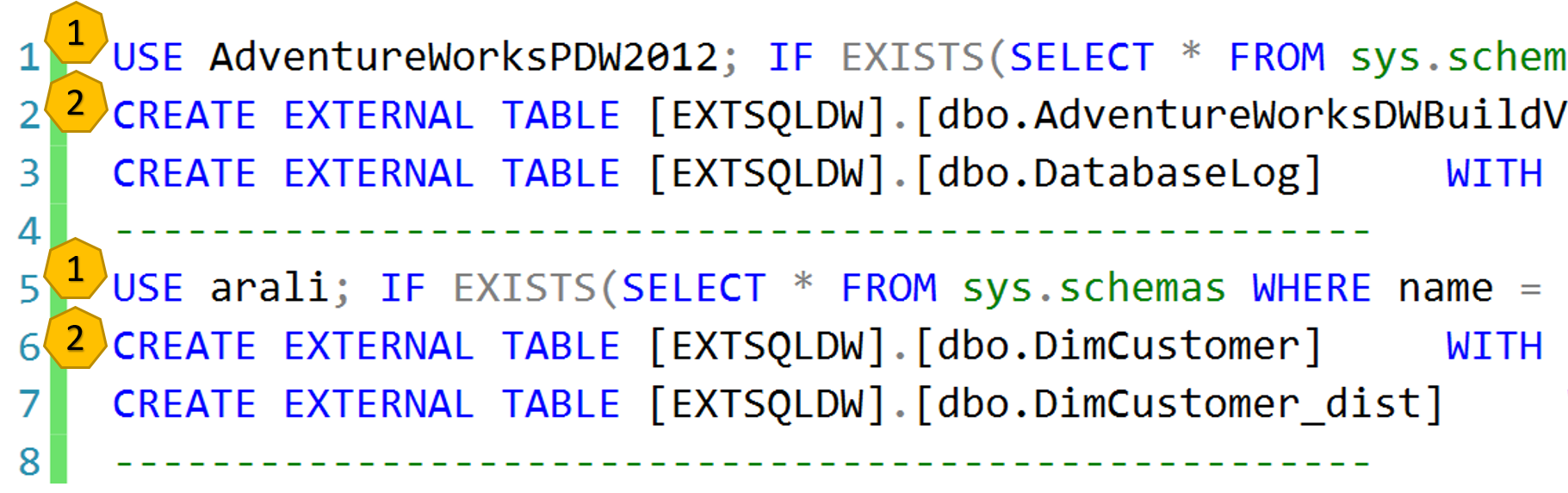


Figure - Export - Script Execution Order

Note - You will need to manually clean up blob storage account before you drop the external table and create it again or if you are executing the script more than once.

Execute **ExportToBlob-Part3.dsql** for cleaning up objects from the APS appliance, again you need to specify few configuration parameters, for example by default this script only deletes external file format and data source (assuming they are not in use); if you want to delete all the external tables created earlier and their corresponding schema, you need to set @DropExternalTableAndSchema to 1 before executing this script.

|  |
| --- |
| USE <AnyUserDatabase>;  DECLARE @DropExternalTableAndSchema BIT = 0  DECLARE @SchemaForExternalTable VARCHAR(255) = 'EXTSQLDW' |

Please note, dropping external tables will not drop files created on blob storage. If you want to delete data as well, you need to delete it from the blob storage account manually.

* 1. Importing Data from Blob Storage to SQL DW

Script available in attachment can be used for automatically generating scripts for creating external tables in SQL DW database. These external tables must point to the same location where data was exported from APS, in the earlier step.

You need to first execute **ImportFromBlob-Part1.dsql** on APS appliance to generate script for external tables (external tables to be created in SQL DW database). The output of the script should look like this (please note, this script might take couple of minutes to hours depending on number of databases, tables and columns, as it does lots of string manipulation for dynamic SQL and string manipulation is slower in APS\SQL DW). The script generated here is in multiple parts copy all the scripts (Part1-Part6). The script is broken down into multiple parts due to limit of 8000 characters for varchar in APS and the table definition could easily exceed that limit:

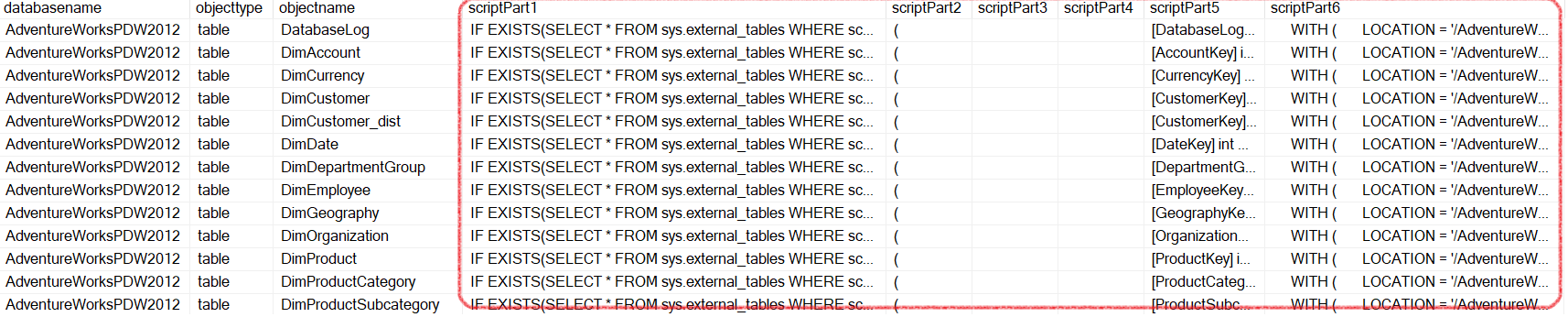


Figure - Import - External Tables

Again, the framework allows you to specify databases to work on. You can specify one or more databases to consider in single execution. For each database you want to include, you need to one INSERT statement for each database, as shown below, in **ImportFromBlob-Part1.dsql** before execution:

|  |
| --- |
| INSERT INTO DatabasesToInclude VALUES ('AdventureWorksPDW2012') |

Next you need to connect to SQL DW database and execute **ImportFromBlob-Part2.dsql** script but before that you need to change few important configuration parameters and copy scripts from above (as shown in Figure 4) for executing it on SQL DW database:

* @AzureStorageAccount - You need to specify the blob storage location where data was exported earlier. It should be exactly same as above.
* @FieldDelimiter – You need to specify same character delimiter what you specified when exporting data. It should be exactly same as above.

|  |
| --- |
| USE <<SQL DW DatabaseName>>;  --step 1: define all parameters  DECLARE @FormatType VARCHAR(100) = 'DELIMITEDTEXT'  DECLARE @FieldDelimiter VARCHAR(10) = '^|^'  DECLARE @DateFormat VARCHAR(12) = 'MM/dd/yyyy'  DECLARE @DataCompression VARCHAR(100) = 'org.apache.hadoop.io.compress.GzipCodec'  DECLARE @AzureStorageAccount VARCHAR(1000) = 'wasbs://<containername>@<accountname>.blob.core.windows.net/'  DECLARE @AzureStorageAccessKey VARCHAR(1000) = '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*' |

Next, you need to execute **ImportFromExternal-Part1.dsql** script. This gives scripts for creating SQL DW internal tables and importing data into them from external tables (example shown in section 1.2.5, 1.2.6, 1.2.7 and 1.2.8). You need to take result set and execute it into other query window (you can split the result set and execute them in parallel across multiple query windows).

* 1. Generating scripts for other objects

Finally, you can execute scripts (**GenerateModuleScript.dsql**, **GenerateNonClusteredIndex.dsql** and **GenerateUserCreatedStatistics.dsql**) for creating additional objects like non-clustered indexes, statistics, modules etc.

Create the stored procedure from the script CreateStatsForAllColumns.sql. To create statistics on all columns in the table with this procedure, simply call the procedure.

1. References

<https://www.microsoft.com/en-us/sql-server/analytics-platform-system>

<https://docs.microsoft.com/en-us/azure/sql-data-warehouse/sql-data-warehouse-overview-what-is>

<https://blogs.msdn.microsoft.com/sqlcat/2017/05/17/azure-sql-data-warehouse-loading-patterns-and-strategies/>

<https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-guide>

1. Appendix
   1. Configure PolyBase Connectivity to External Data

Before you can start exporting data out from APS appliance to Azure Blob Storage account, there are some configuration changes need to be done and appliance services need to be restarted. This link has details on steps:

<https://msdn.microsoft.com/en-us/sql/analytics-platform-system/configure-polybase-connectivity-to-external-data>

<https://msdn.microsoft.com/en-us/sql/analytics-platform-system/use-a-dns-forwarder-to-resolve-non-appliance-dns-names>