Microsoft

Azure Synapse DW - Pool Best Practices & Field Guidance

Prepared by

Data SQL Ninja Engineering Team (askdmjfordmtools@microsoft.com)

Prepared by

Data SQL Ninja Engineering Team

Advanced Solutions Delivery – Data & AI CTO

askdmjfordmtools@microsoft.com

dmjarchitects@microsoft.com

Revision 1

5/8/2020

Disclaimer

The High-Level Architecture, Migration Dispositions and guidelines in this document is developed in consultation and collaboration with Microsoft Corporation technical architects. Because Microsoft must respond to changing market conditions, this document should not be interpreted as an invitation to contract or a commitment on the part of Microsoft.

Microsoft has provided generic high-level guidance in this document with the understanding that MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THE INFORMATION CONTAINED HEREIN.

This document is provided "as-is". Information and views expressed in this document, including URL and other Internet Web site references, may change without notice.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

Note: The detail provided in this document has been harvested as part of a customer engagement sponsored through the Azure Data SQL Ninja Program.

Table of Contents

1 Introduction

This document covers Best Practices and Configurations we should plan for implementing and setting up Azure Synapse Pool / DW.

Customer will have many questions on what Microsoft recommends from their other global customer experience, how to configure and maintain Synapse Pool so we can get best performance and concurrency at the same time.

This document help you pass our field experience to customer and make their environment production ready.

We Discuss and Explain Data Loading, Transformations and Performance Strategies for Synapse SQL Pool.

2 What is Azure Synapse Analytics (Formerly SQL DW)?

Azure Synapse is an analytics service that brings together enterprise data warehousing and Big Data analytics. It gives you the freedom to query data on your terms, using either serverless on-demand or provisioned resources—at scale. Azure Synapse brings these two worlds together with a unified experience to ingest, prepare, manage, and serve data for immediate BI and machine learning needs.

Azure Synapse has four components:

- Synapse SQL: Complete T-SQL based analytics – Generally Available
  - SQL pool (pay per DWU provisioned)
  - SQL on-demand (pay per TB processed) – (Preview)
- Spark: Deeply integrated Apache Spark (Preview)
- Synapse Pipelines: Hybrid data integration (Preview)
- Studio: Unified user experience. (Preview)

2.1 Synapse SQL Pool in Azure Synapse

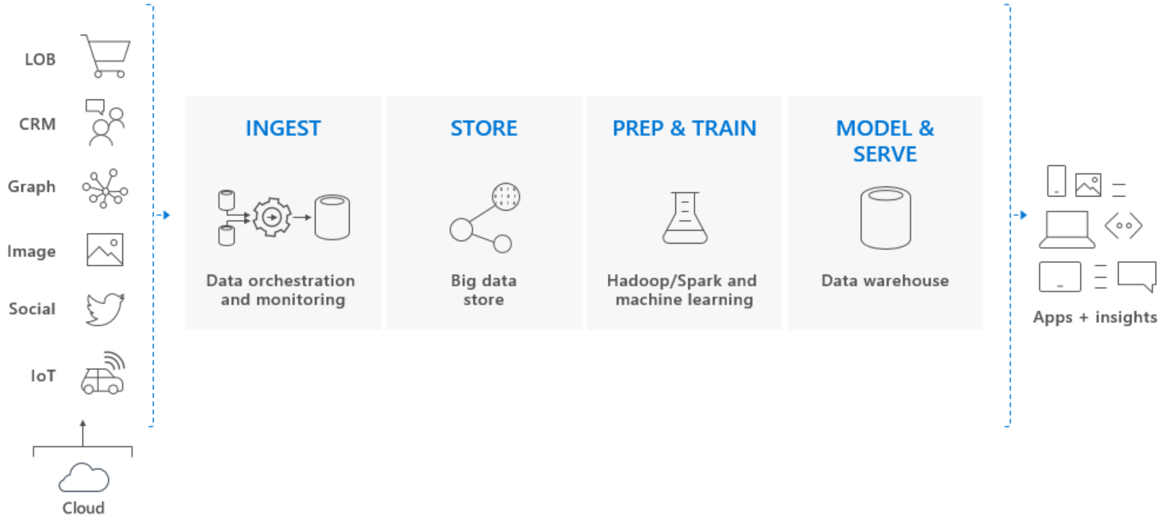
Synapse SQL pool refers to the enterprise data warehousing features that are generally available in Azure Synapse.

SQL pool represents a collection of analytic resources that are being provisioned when using Synapse SQL. The size of SQL pool is determined by Data Warehousing Units (DWU).

Import big data with simple Polybase T-SQL queries, and then use the power of MPP to run high-performance analytics. As you integrate and analyze, Synapse SQL pool will become the single version of truth your business can count on for faster and more robust insights.

2.2 Key Component of a big data solution

Data warehousing is a key component of a cloud-based, end-to-end big data solution.



In a cloud data solution, data is ingested into big data stores from a variety of sources. Once in a big data store, Hadoop, Spark, and machine learning algorithms prepare and train the data. When the data is ready for complex analysis, Synapse SQL pool uses Polybase to query the big data stores. Polybase uses standard T-SQL queries to bring the data into Synapse SQL pool tables.

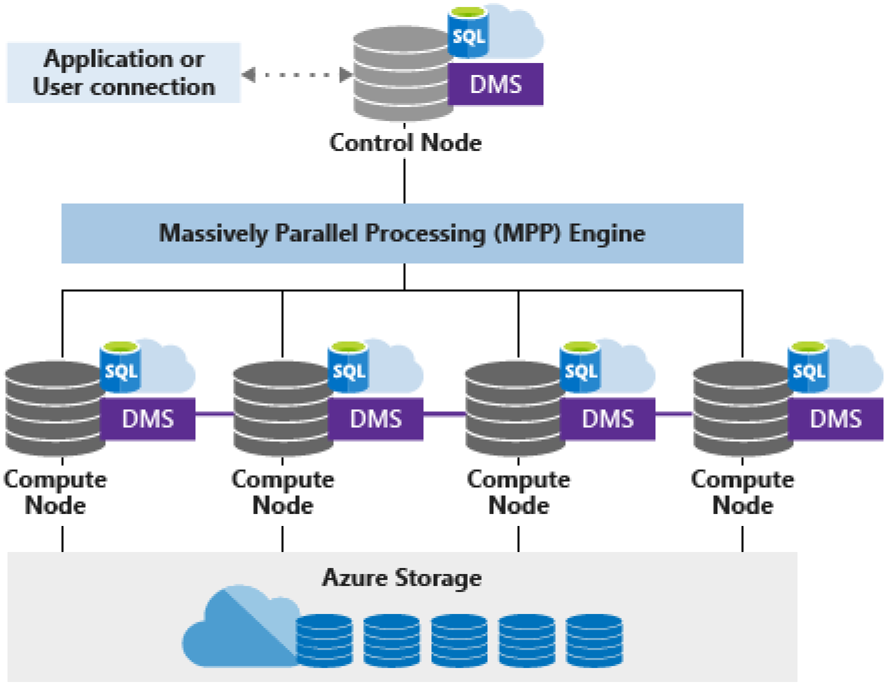
Synapse SQL pool stores data in relational tables with columnar storage. This format significantly reduces the data storage costs and improves query performance. Once data is stored, you can run analytics at massive scale. Compared to traditional database systems, analysis queries finish in seconds instead of minutes, or hours instead of days.

The analysis results can go to worldwide reporting databases or applications. Business analysts can then gain insights to make well-informed business decisions.

3 Azure Synapse Pool Architecture

3.1 Synapse SQL MPP Architecture Components

Synapse SQL leverages a scale-out architecture to distribute computational processing of data across multiple nodes. The unit of scale is an abstraction of compute power that is known as a data warehouse unit. Compute is separate from storage, which enables you to scale compute independently of the data in your system.



SQL Analytics uses a node-based architecture. Applications connect and issue T-SQL commands to a Control node, which is the single point of entry for SQL Analytics. The Control node runs the MPP engine, which optimizes queries for parallel processing, and then passes operations to Compute nodes to do their work in parallel.

The Compute nodes store all user data in Azure Storage and run the parallel queries. The Data Movement Service (DMS) is a system-level internal service that moves data across the nodes as necessary to run queries in parallel and return accurate results.

With decoupled storage and compute, when using Synapse SQL pool one can:

- Independently size compute power irrespective of your storage needs.
- Grow or shrink compute power, within a SQL pool (data warehouse), without moving data.
- Pause compute capacity while leaving data intact, so you only pay for storage.
- Resume compute capacity during operational hours.

3.1.1 Azure Storage

Synapse SQL leverages Azure Storage to keep your user data safe. Since your data is stored and managed by Azure Storage, there is a separate charge for your storage consumption. The data is sharded into **distributions** to optimize the performance of the system. You can choose which sharding pattern to use to distribute the data when you define the table. These sharding patterns are supported:

- Hash
- Round Robin
- Replicate

3.1.2 Control node

The Control node is the brain of the architecture. It is the front end that interacts with all applications and connections. The MPP engine runs on the Control node to optimize and coordinate parallel queries. When you submit a T-SQL query, the Control node transforms it into queries that run against each distribution in parallel.

3.1.3 Compute nodes

The Compute nodes provide the computational power. Distributions map to Compute nodes for processing. As you pay for more compute resources, distributions are remapped to available Compute nodes. The number of compute nodes ranges from 1 to 60 and is determined by the service level for Synapse SQL.

Each Compute node has a node ID that is visible in system views. You can see the Compute node ID by looking for the node_id column in system views whose names begin with sys.pdw_nodes. For a list of these system views, see MPP system views.

3.1.4 Data Movement Service

Data Movement Service (DMS) is the data transport technology that coordinates data movement between the Compute nodes. Some queries require data movement to ensure the parallel queries return accurate results. When data movement is required, DMS ensures the right data gets to the right location.

3.2 Distributions

A distribution is the basic unit of storage and processing for parallel queries that run on distributed data. When SQL Analytics runs a query, the work is divided into 60 smaller queries that run in parallel.

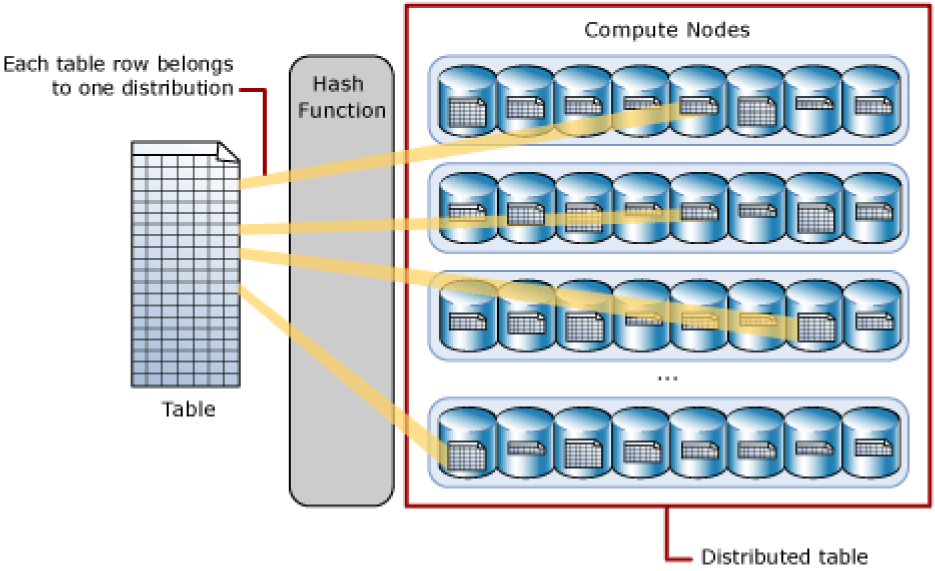
Each of the 60 smaller queries runs on one of the data distributions. Each Compute node manages one or more of the 60 distributions. A SQL pool with maximum compute resources has one distribution per Compute node. A SQL pool with minimum compute resources has all the distributions on one compute node.

3.3 Hash-distributed tables

A hash distributed table can deliver the highest query performance for joins and aggregations on large tables.

To shard data into a hash-distributed table, a hash function is used to deterministically assign each row to one distribution. In the table definition, one of the columns is designated as the distribution column. The hash function uses the values in the distribution column to assign each row to a distribution.

The following diagram illustrates how a full (non-distributed table) gets stored as a hash-distributed table.



- Each row belongs to one distribution.
- A deterministic hash algorithm assigns each row to one distribution.
- The number of table rows per distribution varies as shown by the different sizes of tables.

There are performance considerations for the selection of a distribution column, such as distinctness, data skew, and the types of queries that run on the system.

3.4 Round-robin distributed tables

A round-robin table is the simplest table to create and delivers fast performance when used as a staging table for loads.

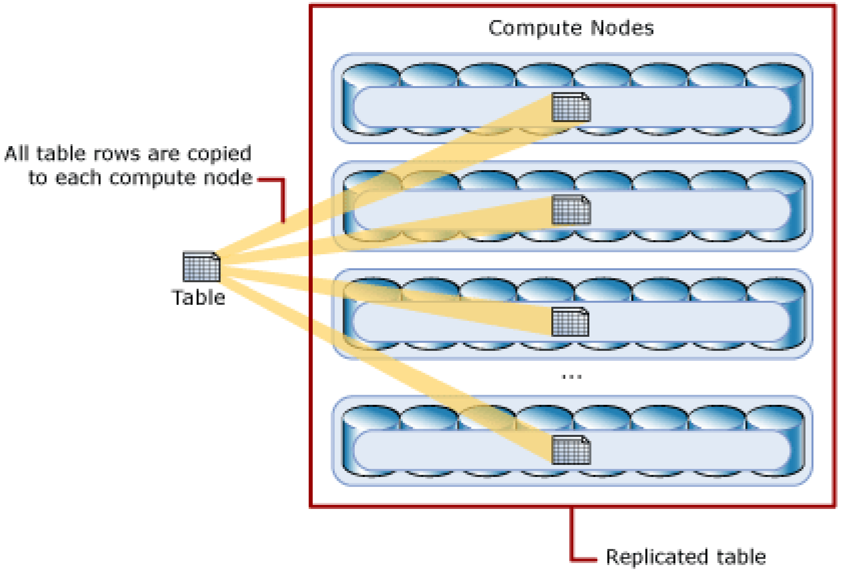
A round-robin distributed table distributes data evenly across the table but without any further optimization. A distribution is first chosen at random and then buffers of rows are assigned to distributions sequentially. It is quick to load data into a round-robin table, but query performance can often be better with hash distributed tables. Joins on round-robin tables require reshuffling data, which takes additional time.

3.5 Replicated Tables

A replicated table provides the fastest query performance for small tables.

A table that is replicated caches a full copy of the table on each compute node. Consequently, replicating a table removes the need to transfer data among compute nodes before a join or aggregation. Replicated tables are best utilized with small tables. Extra storage is required and there is additional overhead that is incurred when writing data, which make large tables impractical.

The diagram below shows a replicated table that is cached on the first distribution on each compute node.



4 When to Recommend Synapse Pool?

Before migrating, you want to be certain SQL Data Warehouse is the right solution for your workload. SQL Data Warehouse is a distributed system, designed to perform analytics on large volumes of data. Migrating to SQL Data Warehouse requires some design changes that are not too hard to understand but might take some time to implement. If your business requires an enterprise-class data warehouse (DW), the benefits are worth the effort. However, if you don't need the power of SQL Data Warehouse, it is more cost-effective to use SQL Server or Azure SQL Database.

4.1 Consider using SQL Data Warehouse when you

- Have one or more Terabytes of data
- Plan to run analytics on large amounts of data
- Need the ability to scale compute and storage
- Want to save costs by pausing compute resources when you don't need them.
- Batch workload
- Set-based queries
- 1billion+ rows

4.2 Don't use SQL Data Warehouse for operational (OLTP) workloads that have

- High frequency reads and writes
- Large numbers of singleton select
- High volumes of single row inserts
- Row by row processing needs
- Incompatible formats (JSON, XML)
- OLTP workload
- Very high concurrency (200+)
- < 1Tb – Total Dataset Size
- Spatial data and algebra

4.3 Typical Data Sources

- Files (CSV, Parquet, etc.)
- Databases (SQL Server, Oracle, etc.)
- Streams

5 What are Data Warehouse Units?

A Synapse SQL pool represents a collection of analytic resources that are being provisioned. Analytic resources are defined as a combination of CPU, memory, and IO.

These three resources are bundled into units of compute scale called Data Warehouse Units (DWUs). A DWU represents an abstract, normalized measure of compute resources and performance.

A change to your service level alters the number of DWUs that are available to the system, which in turn adjusts the performance, and the cost, of your system.

For higher performance, you can increase the number of data warehouse units. For less performance, reduce data warehouse units. Storage and compute costs are billed separately, so changing data warehouse units does not affect storage costs.

Performance for data warehouse units is based on these data warehouse workload metrics:

- How fast a standard SQL pool query can scan a large number of rows and then perform a complex aggregation. This operation is I/O and CPU intensive.
- How fast the SQL pool can ingest data from Azure Storage Blobs or Azure Data Lake. This operation is network and CPU intensive.
- How fast the `CREATE TABLE AS SELECT` T-SQL command can copy a table. This operation involves reading data from storage, distributing it across the nodes of the appliance and writing to storage again. This operation is CPU, IO, and network intensive.

Increasing DWUs:

- Linearly changes performance of the system for scans, aggregations, and CTAS statements
- Increases the number of readers and writers for Polybase load operations
- Increases the maximum number of concurrent queries and concurrency slots.

6 How many DWU do you need?

The ideal number of data warehouse units depends very much on your workload and the amount of data you have loaded into the system.

Steps for finding the best DWU for your workload:

1. Begin by selecting a smaller DWU.

2. Monitor your application performance as you test data loads into the system, observing the number of DWUs selected compared to the performance you observe.
3. Identify any additional requirements for periodic periods of peak activity. Workloads that show significant peaks and troughs in activity may need to be scaled frequently.

SQL pool is a scale-out system that can provision vast amounts of compute and query sizeable quantities of data.

To see its true capabilities for scaling, especially at larger DWUs, we recommend scaling the data set as you scale to ensure that you have enough data to feed the CPUs. For scale testing, we recommend using at least 1 TB.

Create data warehouse Database with SLO

```
CREATE DATABASE mySQLDW
(
  EDITION = 'Datawarehouse'
 ,SERVICE_OBJECTIVE = 'DW1000c'
)
;
```

View current DWU settings

1. Open SQL Server Object Explorer in Visual Studio.
2. Connect to the master database associated with the logical SQL Database server.
3. Select from the sys.database_service_objectives dynamic management view. Here is an example:

```
SELECT  db.name [Database]
,        ds.edition [Edition]
,        ds.service_objective [Service Objective]
FROM    sys.database_service_objectives   AS ds
JOIN    sys.databases                     AS db ON ds.database_id =
db.database_id

;
```

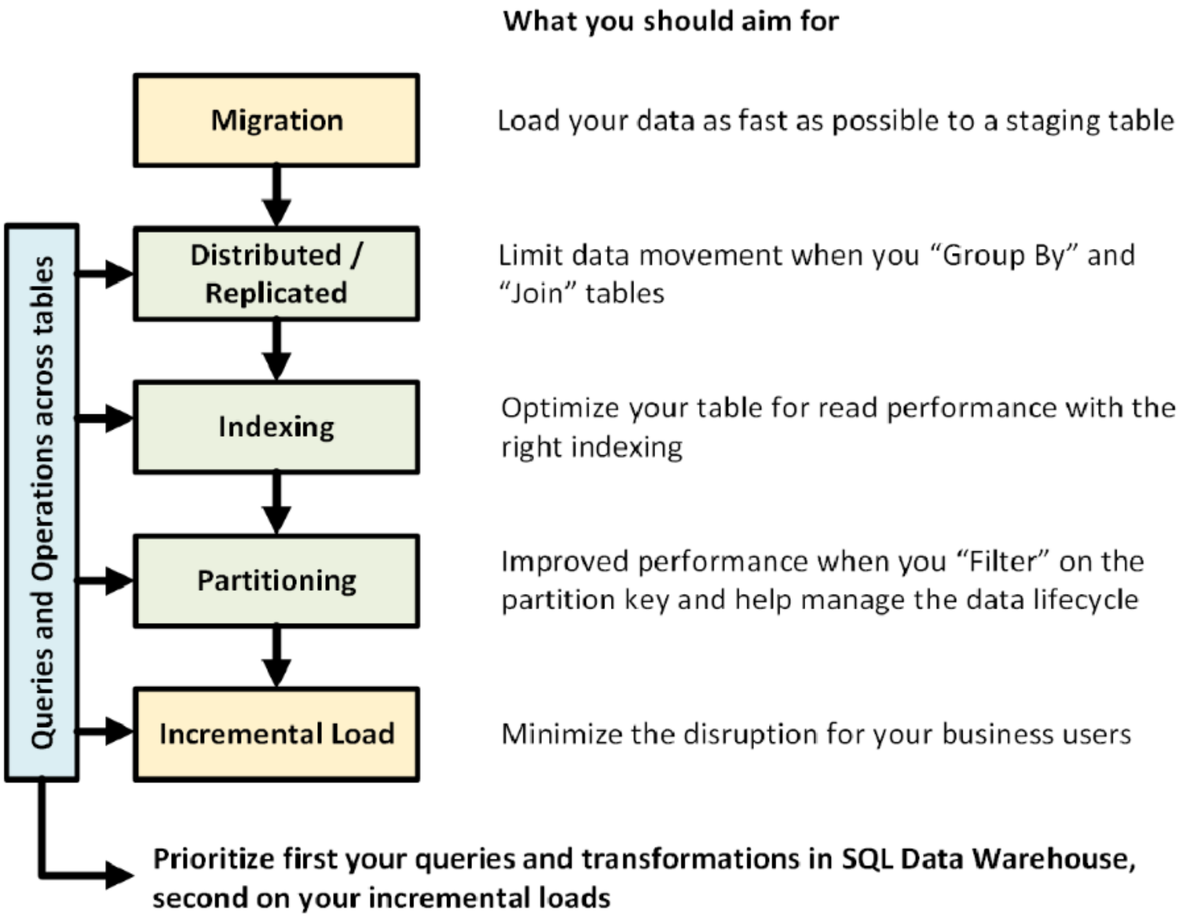
Change data warehouse units

```
ALTER DATABASE MySQLDW
MODIFY (SERVICE_OBJECTIVE = 'DW1000c');
```

7 Best Practices For Landing & Data Loading

This section provides helpful tips and best practices for building Azure Synapse solutions.

The following graphic shows the process of designing a data warehouse:


What you should aim for
Migration
Load your data as fast as possible to a staging table
Distributed / Replicated
Limit data movement when you “Group By” and “Join” tables
Indexing
Optimize your table for read performance with the right indexing
Partitioning
Improved performance when you “Filter” on the partition key and help manage the data lifecycle
Incremental Load
Minimize the disruption for your business users
Queries and Operations across tables
Prioritize first your queries and transformations in SQL Data Warehouse, second on your incremental loads

7.1 ETL VS ELT

Traditional SMP SQL pools use an Extract, Transform, and Load (ETL) process for loading data. Synapse SQL pool, within Azure Synapse Analytics, has a massively parallel processing (MPP) architecture that takes advantage of the scalability and flexibility of compute and storage resources.

Using an Extract, Load, and Transform (ELT) process leverages MPP and eliminates the resources needed for data transformation prior to loading.

While SQL pool supports many loading methods, including popular SQL Server options such as bcp and the SqlBulkCopy API, the fastest and most scalable way to load data is through PolyBase external tables and the COPY statement (preview).

With PolyBase and the COPY statement, you can access external data stored in Azure Blob storage or Azure Data Lake Store via the T-SQL language. For the most flexibility when loading, we recommend using the COPY statement.

7.2 What is ELT?

Extract, Load, and Transform (ELT) is a process by which data is extracted from a source system, loaded into a SQL pool, and then transformed.

The basic steps for implementing ELT are:

1. Extract the source data into text files.
2. Land the data into Azure Blob storage or Azure Data Lake Store.
3. Prepare the data for loading.
4. Load the data into staging tables with PolyBase or the COPY command.
5. Transform the data.
6. Insert the data into production tables.

7.3 Land data into Azure Blob or Azure Data Lake Store

7.3.1 Extract the source data into text files

Getting data out of your source system depends on the storage location. The goal is to move the data into PolyBase, and the COPY supported delimited text or CSV files.

7.3.2 PolyBase and COPY external file formats

With PolyBase and the COPY statement, you can load data from UTF-8 and UTF-16 encoded delimited text or CSV files. In addition to delimited text or CSV files, it loads from the Hadoop file formats such as ORC and Parquet. PolyBase and the COPY statement can also load data from Gzip and Snappy compressed files.

Extended ASCII, fixed-width format, and nested formats such as WinZip or XML aren't supported. If you're exporting from SQL Server, you can use the bcp command-line tool to export the data into delimited text files.

7.3.3 Tools & Services to move data to Azure Storage

To land the data in Azure storage, you can move it to Azure Blob storage or Azure Data Lake Store Gen2. In either location, the data should be stored in text files. PolyBase and the COPY statement can load from either location.

Tools and services you can use to move data to Azure Storage:

- Azure ExpressRoute service enhances network throughput, performance, and predictability. ExpressRoute is a service that routes your data through a dedicated private connection to Azure. ExpressRoute connections do not route data through the public internet. The connections offer more reliability, faster speeds, lower latencies, and higher security than typical connections over the public internet.
- AZCopy utility moves data to Azure Storage over the public internet. This works if your data sizes are less than 10 TB. To perform loads on a regular basis with AZCopy, test the network speed to see if it is acceptable.
- Azure Data Factory (ADF) has a gateway that you can install on your local server. Then you can create a pipeline to move data from your local server up to Azure Storage. To use Data Factory with SQL Analytics, see Loading data for SQL Analytics.

7.3.4 Data Landing from on-prem data sources to ADLS

- Fastest is batch
  - Extract from data source to multiple CSV or Parquet files
  - AzCopy extracts to ADLS
- Alternative is query/insert:
  - Set up SSIS self-hosted integration runtime
  - Use ADF to extract/copy to ADLS
  - Use ADF to execute load procedure

7.3.5 Data Landing from Cloud Data Sources to ADLS

  - Extract using ADF
  - Write to ADLS as Parquet files

- AzCopy is a fast move for files from S3 to ADLS

7.4 Prepare the data for loading & planning

You might need to prepare and clean the data in your storage account before loading. Data preparation can be performed while your data is in the source, as you export the data to text files, or after the data is in Azure Storage. It is easiest to work with the data as early in the process as possible.

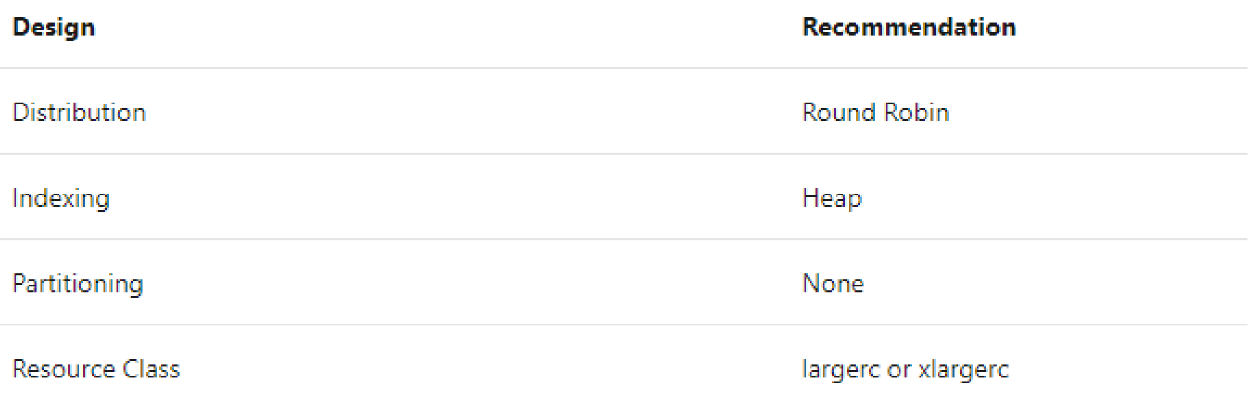
7.4.1 Define external tables

If you are using PolyBase, you need to define external tables in your SQL pool before loading. External tables are not required by the COPY statement. PolyBase uses external tables to define and access the data in Azure Storage.

An external table is similar to a database view. The external table contains the table schema and points to data that is stored outside the SQL pool.

7.4.2 Data migration

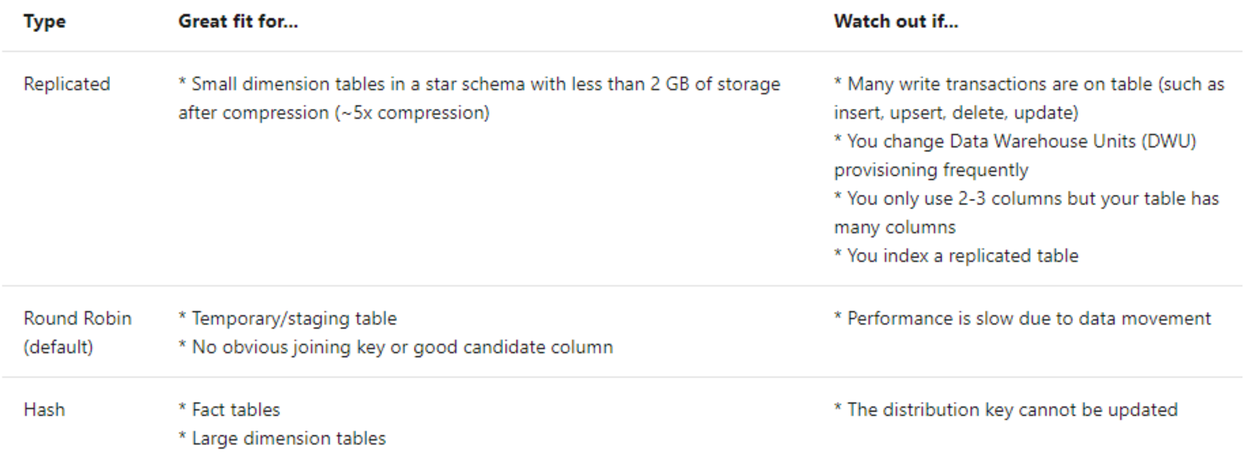
First, load your data into Azure Data Lake Storage or Azure Blob Storage. Next, use PolyBase to load your data into staging tables. Use the following configuration:



Design	Recommendation
Distribution	Round Robin
Indexing	Heap
Partitioning	None
Resource Class	largerc or xlargerc

7.4.3 Distributed or replicated tables

Use the following strategies, depending on the table properties:



Type	Great fit for...	Watch out if...
Replicated	* Small dimension tables in a star schema with less than 2 GB of storage after compression (~5x compression)	* Many write transactions are on table (such as insert, upsert, delete, update) * You change Data Warehouse Units (DWU) provisioning frequently * You only use 2-3 columns but your table has many columns * You index a replicated table
Round Robin (default)	* Temporary/staging table * No obvious joining key or good candidate column	* Performance is slow due to data movement
Hash	* Fact tables * Large dimension tables	* The distribution key cannot be updated

Tips:

- Start with Round Robin but aspire to a hash distribution strategy to take advantage of a massively parallel architecture.
- Make sure that common hash keys have the same data format.
- Don't distribute on varchar format.
- Dimension tables with a common hash key to a fact table with frequent join operations can be hash distributed.
- Use *sys.dm_pdw_nodes_db_partition_stats* to analyze any skewness in the data.
- Use *sys.dm_pdw_request_steps* to analyze data movements behind queries, monitor the time broadcast, and shuffle operations take. This is helpful to review your distribution strategy.

7.4.4 Resource classes

Resource groups are used as a way to allocate memory to queries. If you need more memory to improve query or loading speed, you should allocate higher resource classes. On the flip side, using larger resource classes impacts concurrency. You want to take that into consideration before moving all of your users to a large resource class.

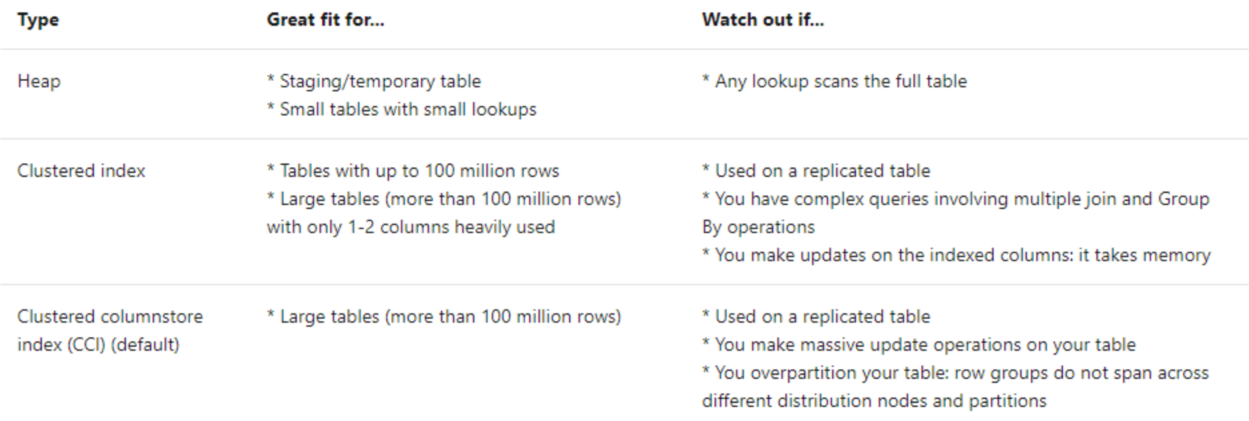
If you notice that queries take too long, check that your users do not run in large resource classes. Large resource classes consume many concurrency slots. They can cause other queries to queue up.

Finally, by using Gen2 of SQL pool, each resource class gets 2.5 times more memory than Gen1.

For Data Loading we recommend Staticrc60 / LargeRC, which will be using good portion of resources for loading, create / Rebuild CCI indexes we should use minimum MediumRC or above.

7.4.5 Index your table

Indexing is helpful for reading tables quickly. There is a unique set of technologies that you can use based on your needs:



Type	Great fit for...	Watch out if...
Heap	* Staging/temporary table * Small tables with small lookups	* Any lookup scans the full table
Clustered index	* Tables with up to 100 million rows * Large tables (more than 100 million rows) with only 1-2 columns heavily used	* Used on a replicated table * You have complex queries involving multiple join and Group By operations * You make updates on the indexed columns: it takes memory
Clustered columnstore index (CCI) (default)	* Large tables (more than 100 million rows)	* Used on a replicated table * You make massive update operations on your table * You overpartition your table: row groups do not span across different distribution nodes and partitions

Tips:

- On top of a clustered index, you might want to add a nonclustered index to a column heavily used for filtering.
- Be careful how you manage the memory on a table with CCI. When you load data, you want the user (or the query) to benefit from a large resource class. Make sure to avoid trimming and creating many small compressed row groups.
- On Gen2, CCI tables are cached locally on the compute nodes to maximize performance.
- For CCI, slow performance can happen due to poor compression of your row groups. If this occurs, rebuild or reorganize your CCI. You want at least 100,000 rows per compressed row groups. The ideal is 1 million rows in a row group.
- Based on the incremental load frequency and size, you want to automate when you reorganize or rebuild your indexes. Spring cleaning is always helpful.
- Be strategic when you want to trim a row group. How large are the open row groups? How much data do you expect to load in the coming days?

7.4.6 Partitioning

You might partition your table when you have a large fact table (greater than 1 billion rows). In 99 percent of cases, the partition key should be based on date. Be careful to not over partition, especially when you have a clustered columnstore index.

With staging tables that require ELT, you can benefit from partitioning. It facilitates data lifecycle management. Be careful not to over partition your data, especially on a clustered columnstore index.

7.5 Ingestion Using Polybase

- SQL Data Warehouse supports many loading methods, including non-PolyBase options (BCP and SQLBulkCopy API), and PolyBase options CTAS/INSERT, PolyBase with SSIS , PolyBase with Azure Databricks and PolyBase with Azure Data Factory (ADF).
- PolyBase is by far the fastest and most scalable SQL Data Warehouse loading method to date, so we recommend it as your default loading mechanism. PolyBase is a scalable, query processing framework compatible with Transact-SQL that can be used to combine and bridge data across relational database management systems, Azure Blob Storage, Azure Data Lake Store and Hadoop database platform ecosystems (APS only)
- As a general rule, we recommend making PolyBase your first choice for loading data into SQL Data Warehouse unless you can’t accommodate PolyBase-supported file formats. Currently PolyBase can load data from UTF-8 and UTF-16 encoded delimited text files as well as the popular Hadoop file formats RC File, ORC, and Parquet (non-nested format). PolyBase can load data from gzip, zlib and Snappy compressed files. PolyBase currently does not support extended ASCII, fixed-file format, WinZip and semi-structured data such as Parquet (nested/hierarchical), JSON, and XML. A popular pattern to load semi-structured data is to use Azure Databricks or similarly HDI/Spark to load the data, flatten/transform to the supported format, then load into SQL DW.

As the following architecture diagrams show, each HDFS bridge of the DMS service from every Compute node can connect to an external resource such as Azure Blob Storage, and then bidirectionally transfer data between SQL Data Warehouse and the external resource.

Another popular pattern is to load into a partitioned aligned stage table via CTAS, then partition switch into the final table.

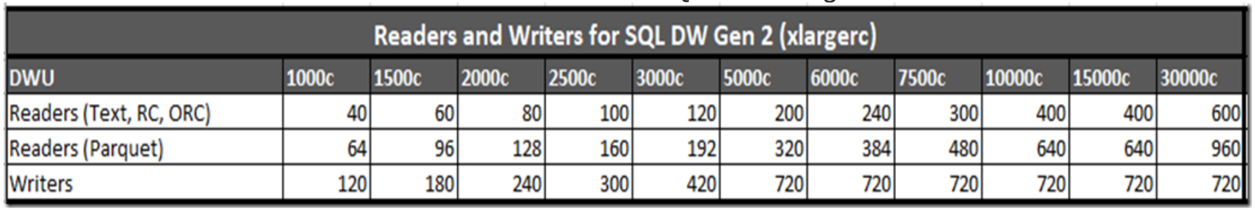
Data Reader, Writer consideration

The number of external readers and writers varies depending on the following factors:

- Gen1 or Gen2 instance
- SLO or size of the instance (DWU)
- Type of operations (DMS query or PolyBase load
- Type of file being loaded (Parquet, text, etc)
- Concurrency at the time the operation was submitted (readers/writers auto-adjust dynamically)

As illustrated in Table 1 below, each DWU has a specific number of readers and writers. As you scale out, each node gets additional number of readers and writers. The static and dynamic resource classes also varies with the number of readers and writers. Note that Parquet files typically has half the number of readers compared to non-Parquet files. The number of readers and writers is an important factor in determining your load performance.

Table 1. Number of readers and writers for Gen 2 SQL DW xlargerc resource class



Readers and Writers for SQL DW Gen 2 (xlargerc)											
DWU	1000c	1500c	2000c	2500c	3000c	5000c	6000c	7500c	10000c	15000c	30000c
Readers (Text, RC, ORC)	40	60	80	100	120	200	240	300	400	400	600
Readers (Parquet)	64	96	128	160	192	320	384	480	640	640	960
Writers	120	180	240	300	420	720	720	720	720	720	720

To check for the number of readers/writers, use the following query (adjust the appropriate request_id and step_index). For more information, see link Monitoring your workload using DMVs

```
SELECT type, count(*) FROM sys.dm_pdw_dms_workers
WHERE request_id = ' QIDXXXX ' AND step_index = XX
group by type;
```

Best practices and considerations when using PolyBase

Here are a few more things to consider when using PolyBase for SQL Data Warehouse loads:

- A single PolyBase load operation provides best performance.
- The load performance scales as you increase DWUs.
- PolyBase automatically parallelizes the data load process, so you don't need to explicitly break the input data into multiple files and issue concurrent loads, unlike some traditional loading practices. Each reader automatically read 512MB for each file for Azure Storage BLOB and 256MB on Azure Data Lake Storage.
- Multiple readers will not work against gzip files. Only a single reader is used per gzip compressed file since uncompressing the file in the buffer is single threaded. Alternatively, generate multiple gzip files. The number of files should be greater than or equal to the total number of readers.
- Multiple readers will work against compressed columnar/block format files (e.g. ORC, RC) since individual blocks are compressed independently.

Known issues when working with different file formats

In addition to the UTF-8/UTF-16 encoding considerations, other known file format issues can arise when using PolyBase.

Mixed intra-file date formats

In a CREATE EXTERNAL FILE FORMAT command, the DATE_FORMAT argument specifies a single format to use for all date and time data in a delimited text file. If the DATE_FORMAT argument isn't designated, the following default formats are used:

DateTime: 'yyyy-MM-dd HH:mm:ss'

- SmallDateTime: 'yyyy-MM-dd HH:mm'
- Date: 'yyyy-MM-dd'
- DateTime2: 'yyyy-MM-dd HH:mm:ss'
- DateTimeOffset: 'yyyy-MM-dd HH:mm:ss'
- Time: 'HH:mm:ss'

For source formats that don't reflect the defaults, you must explicitly specify a custom date format. However, if multiple non-default formats are used within one file, there is currently no method for specifying multiple custom date formats within the PolyBase command.

Fixed-length file format not supported

Fixed-length character file formats—for example, where each column has a fixed width of 10 characters—are not supported today.

If you encounter the restrictions from using PolyBase, considers changing the data extract process to address those limitations. This could be formatting the dates to PolyBase supported format, transforming JSON files to text files, etc. If the option is not possible, then your option is to use any one of the methods in the next section.

7.6 Ingestion Using New Copy Command

This section explains how to use the COPY statement in Azure SQL Data Warehouse for loading from external storage accounts. The COPY statement provides the most flexibility for high-throughput data ingestion into SQL Data Warehouse. Use COPY for the following capabilities:

- Use lower privileged users to load without needing strict CONTROL permissions on the data warehouse
- Execute a single T-SQL statement without having to create any additional database objects
- Properly parse and load CSV files where **delimiters** (string, field, row) **are escaped within string delimited columns**
- Specify a finer permission model without exposing storage account keys using Share Access Signatures (SAS)
- Use a different storage account for the ERRORFILE location (REJECTED_ROW_LOCATION)
- Customize default values for each target column and specify source data fields to load into specific target columns
- Specify a custom row terminator for CSV files
- Leverage SQL Server Date formats for CSV files
- Specify wildcards and multiple files in the storage location path

7.6.1 What is the performance of the COPY command compared to PolyBase?

The COPY command will have better performance depending on your workload. For best loading performance during public preview, consider splitting your input into multiple files when loading CSV.

7.6.2 What is the file splitting guidance for the COPY command loading CSV files?

Guidance on the number of files is outlined in the table below. Once the recommended number of files are reached, you will have better performance the larger the files. For a simple file splitting experience, refer to the following documentation.

DWU	#Files
100	60
200	60
300	60
400	60
500	60
1,000	120
1,500	180
2,000	240
2,500	300
3,000	360
5,000	600
6,000	720

DWU	#Files
7,500	900
10,000	1200
15,000	1800
30,000	3600

7.6.3 What is the file splitting guidance for the COPY command loading Parquet or ORC files?

There is no need to split Parquet and ORC files because the COPY command will automatically split files. Parquet and ORC files in the Azure storage account should be 256MB or larger for best performance.

If you have a single CSV file that is large, since COPY does not have file splits yet, there will be a single reader that will read that file. If you partition file manually (or product multiple files, more than 60), COPY will be able to load in parallel.

We are working on a solution to automatically partition a single big file automatically (just like what Polybase does today) – this should be coming in for GA.

COPY comes with granular permission model, unlike Polybase. COPY will be preferred way of loading data

7.6.4 When will the COPY command be generally available?

The COPY command will be generally available by the end of this calendar year (2020).

7.6.5 Are there any known issues with the COPY command?

- LOB support such as (n)varchar(max) is not available in the COPY statement. This will be available early next year.

7.7 Incremental load

If you're going to incrementally load your data, first make sure that you allocate larger resource classes to loading your data. This is particularly important when loading into tables with clustered columnstore indexes. See resource classes for further details.

We recommend using PolyBase and ADF V2 for automating your ELT pipelines into your data warehouse.

For a large batch of updates in your historical data, consider using a CTAS to write the data you want to keep in a table rather than using INSERT, UPDATE, and DELETE.

8 Best Practices For Data Transformation

While data is in the staging table, perform transformations that your workload requires. Then move the data into a production table.

8.1 Typical Transformations

- Create persistent staging area / data vault
- Standardize data from different sources
- Remove duplicate rows
- Calculate derived values
- Prepare data for facts and dimensions

8.2 Distributions

- Hash distribute large tables
    - Hash has two objectives
        - Balance data across nodes
        - Collocate data for optimised queries
- The best hash key is not necessarily the primary key
- Replicate common tables, especially dimensions – size < 5 GB
- Avoid round_robin, except for very small tables

8.3 Data Movement

- Data movement occurs when a query needs data from other nodes
- Data must be broadcast or shuffled across nodes before the query can execute. This takes time.
- Appropriate hash distribution or replication avoids data movement
- Use the *EXPLAIN* plan or *sys.dm_pdw_request_steps* to identify data movement

8.4 Data Skew

- Data skew means that the rows of a table are not balanced across all nodes.
- Skew means that most of the work for a query takes place on only a few nodes, leading to slow performance because balanced parallel processing cannot be achieved
- Use *dbcc pdw_showspaceused* to identify skew

8.5 Transformations Summary

- Do not Join table types and on columns, which creates incompatible joins

- Incompatible joins are expensive and will cause huge Data Movement
- Avoid Update, Delete Commands and use CTAS instead if updating more than 10% of the records
- 1 million Rows per Batch and Around 60 Million Per Partition
- For incremental loads performance check on the health of the CCI Index
- Failure to maintain current statistics will lead to bad performance

9 Best Practices for Query Presentation Layer

9.1 Result Set Caching

- Cache the results of a query in DW storage. This enables interactive response times for repetitive queries against tables with infrequent data changes.
- The result-set cache persists even if a data warehouse is paused and resumed later.
- Query cache is invalidated and refreshed when underlying table data or query code changes.
- Result cache is evicted regularly based on a time-aware least recently used algorithm (TLRU).

9.2 Materialized Views

- Indexed views cache the schema and data for a view in DW remote storage. They are useful for improving the performance of 'SELECT' statement queries that include aggregations
- Indexed views are automatically updated when data in underlying tables are changed. This is a synchronous operation that occurs as soon as the data is changed.
- The auto caching functionality allows SQL DW Query Optimizer to consider using indexed view even if the view is not referenced in the query.
- Supported aggregations: MAX, MIN, AVG, COUNT, COUNT_BIG, SUM, VAR, STDEV

9.3 Ordered CCI

- Queries against tables with ordered columnstore segments can take advantage of improved segment elimination to drastically reduce the time needed to service a query.
- Columnstore Segments are automatically updated as data is inserted, updated, or deleted in data warehouse tables.

9.4 Workload Group Isolation & Importance

- Total of MIN_RESOURCE_PERCENTAGE of all workload groups cannot exceed 100
- MIN_RESOURCE_PERCENTAGE is reservation, and this shrink resources from another workload group.
- Concurrency range for a workload group is implied with
- CAP_PERCENTAGE_RESOURCE / REQUEST_MIN_RESOURCE_GRANT_PERCENT (or REQUEST_MAX_RESOURCE_GRANT_PERCENT)
- Access the actual business requirement, type of workload and loading / reporting window to design workload isolation and importance.

10 Best Practices for Performance & Maintenance

10.1 Hash distribute large tables

By default, tables are Round Robin distributed. This makes it easy for users to get started creating tables without having to decide how their tables should be distributed. Round Robin tables may perform sufficiently for some workloads, but in most cases selecting a distribution column will perform much better.

The most common example of when a table distributed by a column will far outperform a Round Robin table is when two large fact tables are joined.

For example, if you have an orders table, which is distributed by order_id, and a transactions table, which is also distributed by order_id, when you join your orders table to your transactions table on order_id, this query becomes a pass-through query, which means we eliminate data movement operations. Fewer steps mean a faster query. Less data movement also makes for faster queries.

When loading a distributed table, be sure that your incoming data is not sorted on the distribution key as this will slow down your loads.

10.2 Do not over-partition

While partitioning data can be effective for maintaining your data through partition switching or optimizing scans by with partition elimination, having too many partitions can slow down your queries. Often a high granularity partitioning strategy, which may work well on SQL Server may not work well in SQL pool.

Having too many partitions can also reduce the effectiveness of clustered columnstore indexes if each partition has fewer than 1 million rows. Keep in mind that behind the scenes, SQL pool partitions your data for you into 60 databases, so if you create a table with 100 partitions, this actually results in 6000 partitions.

Each workload is different, so the best advice is to experiment with partitioning to see what works best for your workload. Consider lower granularity than what may have worked for you in SQL Server. For example, consider using weekly or monthly partitions rather than daily partitions.

- Partitions can be useful when maintaining current rows in very large fact tables. Partition switching is a good alternative to full CTAS
- Partitioning CCIs is only useful when the row count is greater than 60million * #partitions

10.3 Too Many Indexes

- Start without indexes. The overhead of maintaining them can be greater than their value.
- A non-clustered index may improve performance of joins when fact tables are joined to very large (billion+) dimensions

10.4 Maintain statistics

SQL pool can be configured to automatically detect and create statistics on columns. The query plans created by the optimizer are only as good as the available statistics.

We recommend that you enable AUTO_CREATE_STATISTICS for your databases and keep the statistics updated daily or after each load to ensure that statistics on columns used in your queries are always up to date.

If you find it is taking too long to update all of your statistics, you may want to try to be more selective about which columns need frequent statistics updates. For example, you might want to update date columns, where new values may be added, daily.

You will gain the most benefit by having updated statistics on columns involved in joins, columns used in the WHERE clause and columns found in GROUP BY.

Until auto-statistics are generally available, manual maintenance of statistics is required. It's important to update statistics as *significant* changes happen to your data. This helps optimize your query plans. If you find that it takes too long to maintain all of your statistics, be more selective about which columns have statistics.

You can also define the frequency of the updates. For example, you might want to update date columns, where new values might be added, on a daily basis. You gain the most benefit by having statistics on columns involved in joins, columns used in the WHERE clause, and columns found in GROUP BY.

10.5 Group INSERT statements into batches

A one-time load to a small table with an INSERT statement or even a periodic reload of a look-up may perform well for your needs with a statement like `INSERT INTO MyLookup VALUES (1, 'Type 1')`.

However, if you need to load thousands or millions of rows throughout the day, you might find that singleton INSERTS just can't keep up. Instead, develop your processes so that they write to a file and another process periodically comes along and loads this file.

10.6 Minimize transaction sizes

INSERT, UPDATE, and DELETE statements run in a transaction and when they fail they must be rolled back. To minimize the potential for a long rollback, minimize transaction sizes whenever possible. This can be done by dividing INSERT, UPDATE, and DELETE statements into parts.

For example, if you have an INSERT that you expect to take 1 hour, if possible, break up the INSERT into four parts, which will each run in 15 minutes. Leverage special Minimal

Logging cases, like CTAS, TRUNCATE, DROP TABLE, or INSERT to empty tables, to reduce rollback risk.

Another way to eliminate rollbacks is to use Metadata Only operations like partition switching for data management. For example, rather than execute a DELETE statement to delete all rows in a table where the order_date was in October of 2001, you could partition your data monthly and then switch out the partition with data for an empty partition from another table (see ALTER TABLE examples).

For unpartitioned tables, consider using a CTAS to write the data you want to keep in a table rather than using DELETE. If a CTAS takes the same amount of time, it is a much safer operation to run as it has minimal transaction logging and can be canceled quickly if needed.

See also Understanding transactions, Optimizing transactions, Table partitioning, TRUNCATE TABLE, ALTER TABLE, and Create table as select (CTAS).

10.7 Reduce query result sizes

This step helps you avoid client-side issues caused by large query result. You can edit your query to reduce the number of rows returned. Some query generation tools allow you to add "top N" syntax to each query. You can also CETAS the query result to a temporary table and then use PolyBase export for the down-level processing.

10.8 Use the smallest possible column size

When defining your DDL, using the smallest data type that will support your data will improve query performance. This approach is particularly important for CHAR and VARCHAR columns.

If the longest value in a column is 25 characters, then define your column as VARCHAR (25). Avoid defining all character columns to a large default length. In addition, define columns as VARCHAR when that is all that is needed rather than use NVARCHAR

10.9 Use temporary heap tables for transient data

When you are temporarily landing data, you may find that using a heap table will make the overall process faster. If you are loading data only to stage it before running more transformations, loading the table to heap table will be much faster than loading the data to a clustered columnstore table.

In addition, loading data to a temp table will also load much faster than loading a table to permanent storage. Temporary tables start with a "#" and are only accessible by the session that created it, so they may only work in limited scenarios.

Heap tables are defined in the WITH clause of a CREATE TABLE. If you do use a temporary table, remember to create statistics on that temporary table too.

10.10 Optimize clustered columnstore tables

Clustered columnstore indexes are one of the most efficient ways you can store your data in SQL pool. By default, tables in SQL pool are created as Clustered ColumnStore. To get the best performance for queries on columnstore tables, having good segment quality is important.

When rows are written to columnstore tables under memory pressure, columnstore segment quality may suffer. Segment quality can be measured by number of rows in a compressed Row Group. See the Causes of poor columnstore index quality in the Table indexes article for step-by-step instructions on detecting and improving segment quality for clustered columnstore tables.

Because high-quality columnstore segments are important, it's a good idea to use users IDs that are in the medium or large resource class for loading data. Using lower data warehouse units means you want to assign a larger resource class to your loading user.

Since columnstore tables generally won't push data into a compressed columnstore segment until there are more than 1 million rows per table and each SQL pool table is partitioned into 60 tables, as a rule of thumb, columnstore tables won't benefit a query unless the table has more than 60 million rows. For table with less than 60 million rows, it may not make any sense to have a columnstore index. It also may not hurt.

Furthermore, if you partition your data, then you will want to consider that each partition will need to have 1 million rows to benefit from a clustered columnstore index. If a table has 100 partitions, then it will need to have at least 6 billion rows to benefit from a clustered columns store (60 distributions *100 partitions* 1 million rows).

If your table does not have 6 billion rows in this example, either reduce the number of partitions or consider using a heap table instead. It also may be worth experimenting to see if better performance can be gained with a heap table with secondary indexes rather than a columnstore table.

When querying a columnstore table, queries will run faster if you select only the columns you need.

10.11 Use larger resource class to improve query performance

SQL pool uses resource groups as a way to allocate memory to queries. Out of the box, all users are assigned to the small resource class, which grants 100 MB of memory per distribution. Since there are always 60 distributions and each distribution is given a minimum of 100 MB, system wide the total memory allocation is 6,000 MB, or just under 6 GB.

Certain queries, like large joins or loads to clustered columnstore tables, will benefit from larger memory allocations. Some queries, like pure scans, will yield no benefit. However, utilizing larger resource classes reduces concurrency, so you will want to take this impact into consideration before moving all of your users to a large resource class.

See also Resource classes for workload management.

10.12 Use Smaller Resource Class to Increase Concurrency

If you notice that user queries seem to have a long delay, it could be that your users are running in larger resource classes and are consuming many concurrency slots causing other queries to queue up. To see if users queries are queued, run `SELECT * FROM sys.dm_pdw_waits` to see if any rows are returned.

See also Resource classes for workload management, sys.dm_pdw_waits

10.13 Queries and operations across tables

When you know in advance the primary operations and queries to be run in your data warehouse, you can prioritize your data warehouse architecture for those operations. These queries and operations might include:

- Joining one or two fact tables with dimension tables, filtering the combined table, and then appending the results into a data mart.
- Making large or small updates into your fact sales.
- Appending only data to your tables.

Knowing the types of operations in advance helps you optimize the design of your tables

10.14 Tune query performance with new product enhancements

- Performance tuning with materialized views
- Performance tuning with ordered clustered columnstore index
- Performance tuning with result set caching

10.15 Use PolyBase to load and export data quickly

SQL pool supports loading and exporting data through several tools including Azure Data Factory, PolyBase, and BCP. For small amounts of data where performance isn't critical, any tool may be sufficient for your needs. However, when you are loading or exporting large volumes of data or fast performance is needed, PolyBase is the best choice.

PolyBase is designed to leverage the MPP (Massively Parallel Processing) architecture and will load and export data magnitudes faster than any other tool. PolyBase loads can be run using CTAS or INSERT INTO.

Using CTAS will minimize transaction logging and the fastest way to load your data.

Azure Data Factory also supports PolyBase loads and can achieve similar performance as CTAS. PolyBase supports a variety of file formats including Gzip files.

To maximize throughput when using gzip text files, break up files into 60 or more files to maximize parallelism of your load. For faster total throughput, consider loading data concurrently.

Load data,
Guide for using PolyBase
SQL pool loading patterns and strategies

Load Data with Azure Data Factory
Move data with Azure Data Factory
CREATE EXTERNAL FILE FORMAT
Create table as select (CTAS).

10.16 Load then query external tables

While Polybase, also known as external tables, can be the fastest way to load data, it is not optimal for queries. Polybase tables currently only support Azure blob files and Azure Data Lake storage. These files do not have any compute resources backing them.

As a result, SQL pool cannot offload this work and therefore must read the entire file by loading it to tempdb to read the data. Therefore, if you have several queries that will be querying this data, it is better to load this data once and have queries use the local table.

10.17 Use DMVs to monitor and optimize your queries

SQL pool has several DMVs that can be used to monitor query execution. The Monitor your workload using DMVs article details step-by-step instructions on how to look at the details of an executing query.

To quickly find queries in these DMVs, using the LABEL option with your queries can help.

Monitor your workload using DMVs,
LABEL,
OPTION,
sys.dm_exec_sessions
sys.dm_pdw_exec_requests
sys.dm_pdw_request_steps
sys.dm_pdw_sql_requests
sys.dm_pdw_dms_workers
DBCC PDW_SHOWEXECUTIONPLAN
sys.dm_pdw_waits

10.18 Lower your cost

A key feature of Azure Synapse is the ability to manage compute resources. You can pause SQL pool when you're not using it, which stops the billing of compute resources. You can scale resources to meet your performance demands. To pause, use the Azure portal or PowerShell. To scale, use the Azure portal, PowerShell, T-SQL, or a REST API.

11 Further References

- Data loading strategies for Synapse SQL pool
- Azure SQL Data Warehouse loading patterns and strategies
- Load data into Azure SQL Data Warehouse by using Azure Data Factory
- Tutorial: Extract, transform, and load data by using Azure Databricks
- Indexing tables in Synapse SQL pool
- Table statistics in Synapse SQL pool
- How to monitor the Gen2 cache
- Common Data Warehouse Functionality Redefined By CTAS
- Near real-time analytics in Azure SQL Data Warehouse
- Monitoring resource utilization and query activity in Azure Synapse Analytics
- Troubleshooting SQL Analytics in Azure Synapse
- Monitor your Azure Synapse Analytics SQL pool workload using DMVs

12 Feedback and suggestions

If you have feedback or suggestions for improving this data migration asset, please contact the Data Migration Jumpstart Team (askdmjfordmtools@microsoft.com). Thanks for your support!

Note: For additional information about migrating various source databases to Azure, see the Azure Database Migration Guide.