Table of Contents

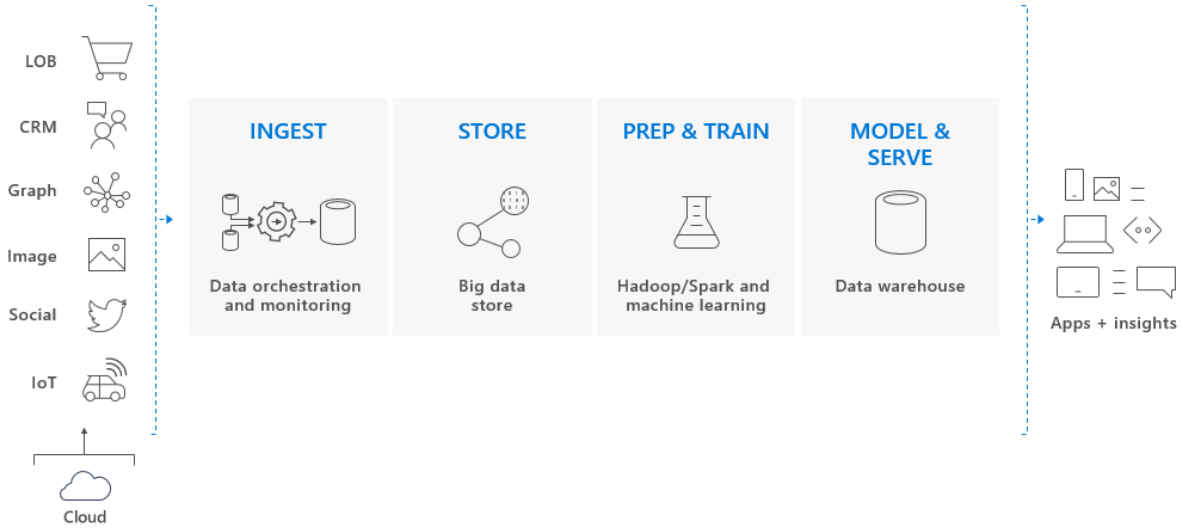
What is Azure SQL Data Warehouse?

12/4/2017 • 2 min to read • Edit Online

SQL Data Warehouse is a cloud-based Enterprise Data Warehouse (EDW) that leverages Massively Parallel Processing (MPP) to quickly run complex queries across petabytes of data. Use SQL Data Warehouse as a key component of a big data solution. Import big data into SQL Data Warehouse with simple PolyBase T-SQL queries, and then use the power of MPP to run high-performance analytics. As you integrate and analyze, the data warehouse will become the single version of truth your business can count on for insights.

Key component of big data solution

SQL Data Warehouse is a key component of an end-to-end big data solution in the Cloud.



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, SQL Data Warehouse uses PolyBase to query the big data stores. PolyBase uses standard T-SQL queries to bring the data into SQL Data Warehouse.

SQL Data Warehouse stores data into relational tables with columnar storage. This format significantly reduces the data storage costs, and improves query performance. Once data is stored in SQL Data Warehouse, 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.

Optimization choices

SQL Data Warehouse offers performance tiers designed for flexibility to meet your data needs, whether big or small. You can choose a data warehouse that is optimized for elasticity or for compute.

- The **Optimized for Elasticity performance tier** separates the compute and storage layers in the architecture. This option excels on workloads that can take full advantage of the separation between compute and storage by scaling frequently to support short periods of peak activity. This compute tier has the lowest entry price point and scales to support the majority of customer workloads.

Solid State Disk cache that keeps the most frequently accessed data close to the CPUs, which is exactly where you want it. By automatically tiering the storage, this performance tier excels with complex queries since all IO is kept local to the compute layer. Furthermore, the columnstore is enhanced to store an unlimited amount of data in your SQL Data Warehouse. The Optimized for Compute performance tier provides the greatest level of scalability, enabling you to scale up to 30,000 compute Data Warehouse Units (cDWU). Choose this tier for workloads that requires continuous, blazing fast, performance.

Next steps

Now that you know a bit about SQL Data Warehouse, learn how to quickly create a SQL Data Warehouse and load sample data. If you are new to Azure, you may find the Azure glossary helpful as you encounter new terminology. Or look at some of these other SQL Data Warehouse Resources.

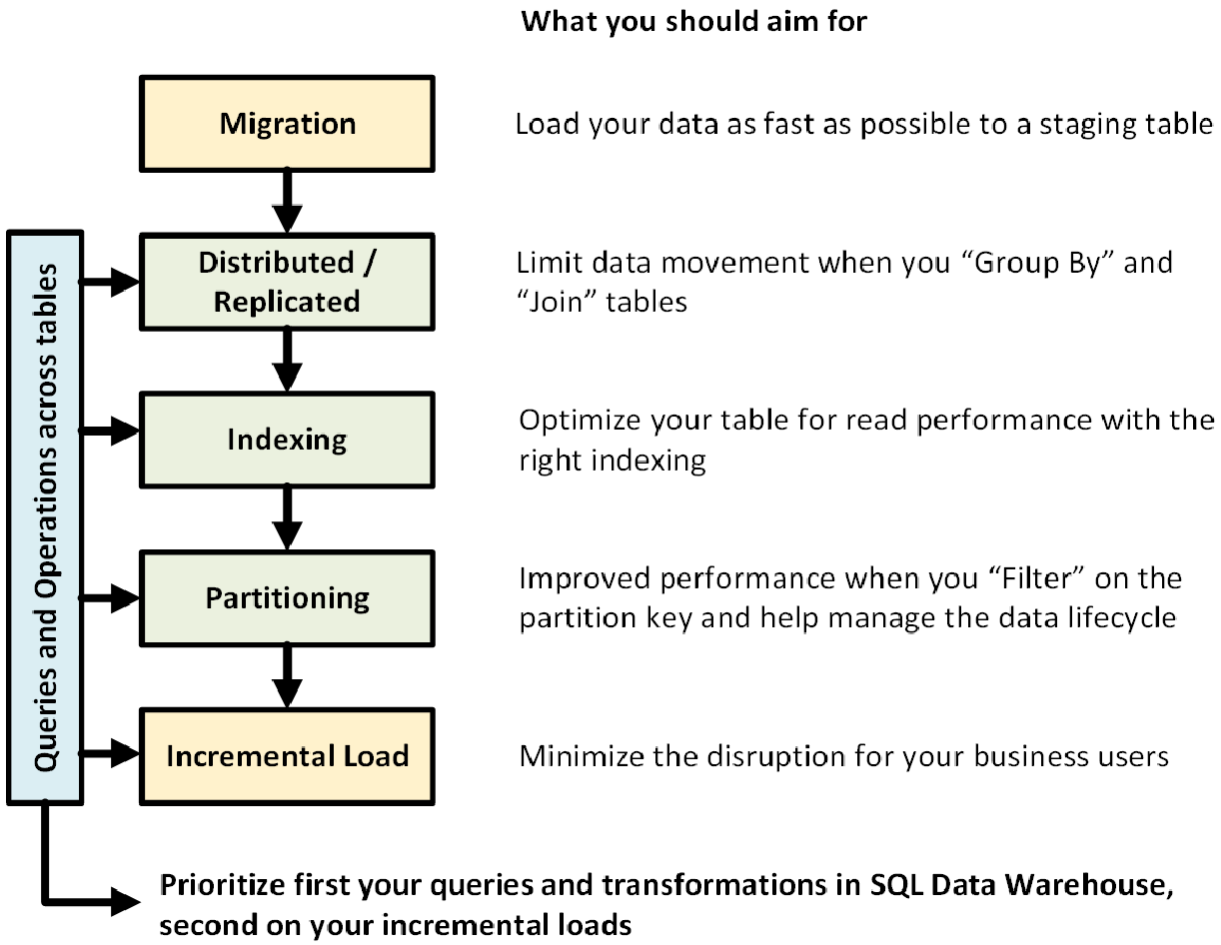
- Customer success stories
- Blogs
- Feature requests
- Videos
- Customer Advisory Team blogs
- Create support ticket
- MSDN forum
- Stack Overflow forum
- Twitter

Cheat sheet for Azure SQL Data Warehouse

1/12/2018 • 6 min to read • Edit Online

This page should help you design for the main use cases your data warehouse solution. This cheat sheet should be a great support in your journey to build a world-class data warehouse but we highly recommend learning more about each step in details. First, we recommend reading this great article about what SQL Data Warehouse **is and is not**.

Following is a sketch of the process you should follow when starting to design SQL Data Warehouse.



Queries and Operations across tables

It is truly important to know in advance the most important operations and queries taking place in your data warehouse. Your data warehouse architecture should be prioritized for those operations. Common examples of operations could be:

- Joining one or two fact tables with dimension tables, filtering this table for a period of time and append the results into a data mart
- Making large or small updates into your fact sales
- Appending only data to your tables

Knowing the type of operations helps you optimize the design of your tables.

Data Migration

We recommend to first load your data **into ADLS** or Azure Blob Storage. Then, you should use Polybase to load your data into SQL Data Warehouse in a staging table. We recommend the following configuration:

DESIGN	RECOMMENDATION
Distribution	Round Robin
Indexing	Heap
Partitioning	None
Resource Class	largerc or xlargerc

Learn more about **data migration**, **data loading** with **deeper guidance on loading**.

Distributed or Replicated Tables

We recommend 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 on table (e.g.: insert, upsert, delete, update) • Change Data Warehouse Units (DWU) provisioning frequently • You only use 2-3 columns and your table has many columns • You index a replicated table
Round Robin (Default)	• Temporary/Staging table • No obvious joining key or good candidate column	• Slow performance due to data movement
Hash	• Fact tables • Large dimension tables	• The distribution key cannot be updated

Tips:

- Start with Round Robin but aspire for a hash distribution strategy to take advantage of a Massive Parallel Architecture
- Make sure that common hash keys have the same data format
- Don't distribute on varchar format
- Dimension tables with common hash key to a fact table with frequent join operations could 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. Helpful to review your distribution strategy.

Learn more about **replicated tables** and **distributed tables**.

Indexing your table

Indexing is **great** for reading quickly tables. There is a unique set of technologies 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	• Up to 100-m rows table • Large tables (more than 100-m rows) with only 1-2 columns are heavily used	• Used on a replicated table • Complex queries involving multiple Join, Group By operations • Make updates on the indexed columns, it takes memory
Clustered Columnstore Index (CCI) (Default)	• Large tables (more than 100-m rows)	• Used on a replicated table • You make massive update operations on your table • Over-partition 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 Nonclustered Index to a column heavily used for filter.
- 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. You make sure to avoid trimming and creating many small compressed row groups
- Optimized for Compute Tier rocks with CCI
- For CCI, slow performance can happen due to poor compression of your row groups, you might want to rebuild or reorganize your CCI. You want at least 100k rows per compressed Row Groups. The ideal is 1-m 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?

Learn more about **indexes**.

Partitioning

You might partition your table when you have a large fact tables (>1B row table). 99% of the 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 ETL, you can benefit from partitioning. It facilitates data lifecycle management. Be careful not to overpartition your data, especially on a Clustered Columnstore Index.

Learn more about **partitions**.

Incremental load

First, you should make sure that you allocate larger resource classes to loading your data. We recommend using Polybase and ADF V2 for automating your ETL pipelines into SQL DW.

For a large batch of updates in your historical data, we recommend deleting first the concerned data. Then you can make a bulk insert of the new data. This 2-step approach is more efficient.

Maintain statistics

Auto-statistics are going to be Generally Available soon. Until then, SQL Data Warehouse requires manual

optimize your query plans. If you find it takes too long to maintain all of your statistics, you may want to 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 may be added, daily. 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.

Learn more about **statistics**.

Resource class

SQL Data Warehouse uses resource groups 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 it 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 if you use the Compute Optimized Tier, each resource class gets 2.5x more memory than on the Elastic Optimized Tier.

Learn more how to work with **resource classes and concurrency**.

Lower your cost

A key feature of SQL Data Warehouse is the ability to pause when you are not using it, which stops the billing of compute resources. Another key feature is the ability to scale resources. Pausing and Scaling can be done via the Azure portal or through PowerShell commands.

Auto-scale now at the time you want with Azure Functions:

Optimize you architecture for performance

We recommend considering SQL database and Azure Analysis Services in a Hub and Spokes architecture. That solution can provide workload isolation between different user groups while also leveraging some advanced security features from SLQ DB and Azure Analysis Services. This is also a way to provide limitless concurrency to your users.

Learn more about **typical architectures leveraging SQL DW**.

Deploy in a click your spokes in SQL DB databases from SQL DW:

Create and query an Azure SQL data warehouse in the Azure portal

11/28/2017 • 6 min to read • Edit Online

Quickly create and query an Azure SQL data warehouse using the Azure portal.

If you don't have an Azure subscription, create a free account before you begin.

Before you begin

Download and install the newest version of SQL Server Management Studio (SSMS).

Sign in to the Azure portal

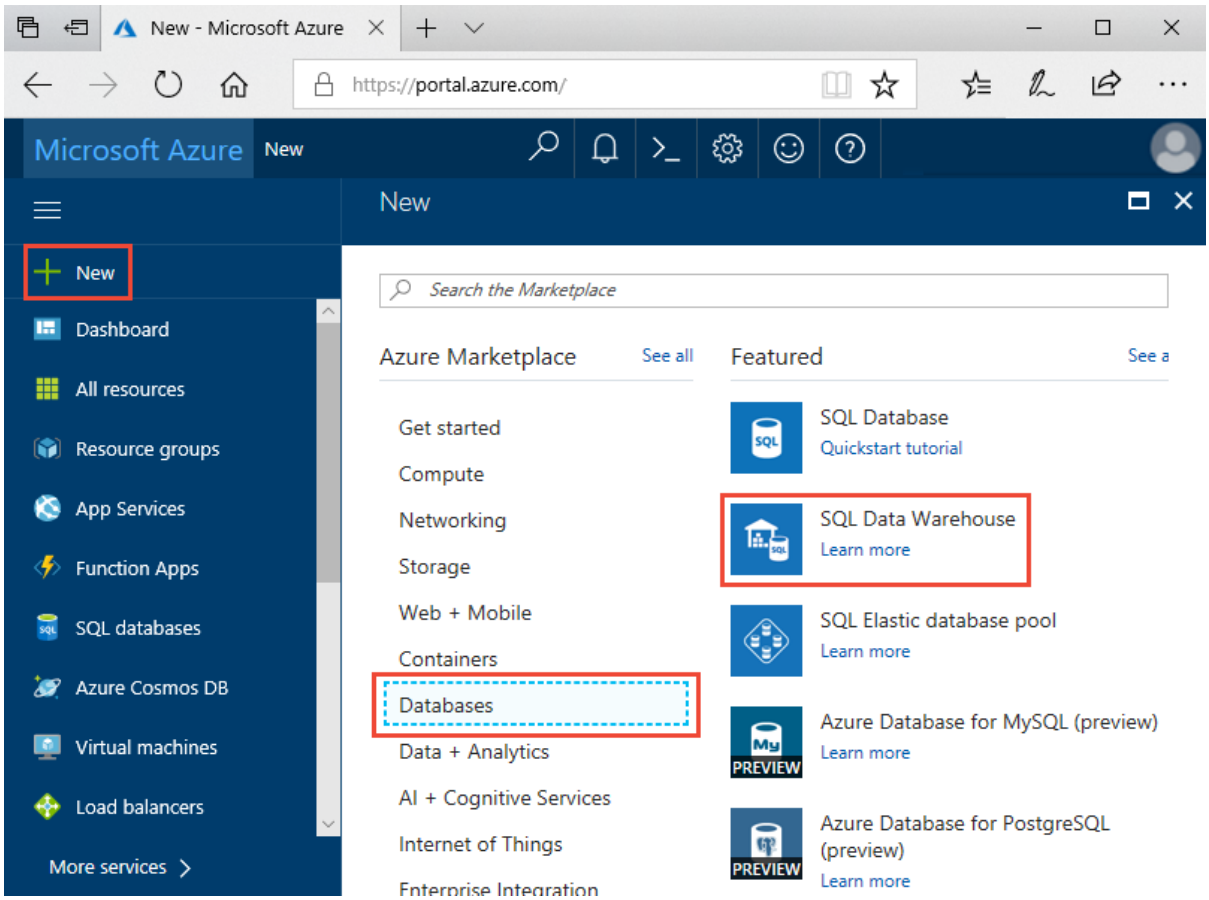
Sign in to the Azure portal.

Create a data warehouse

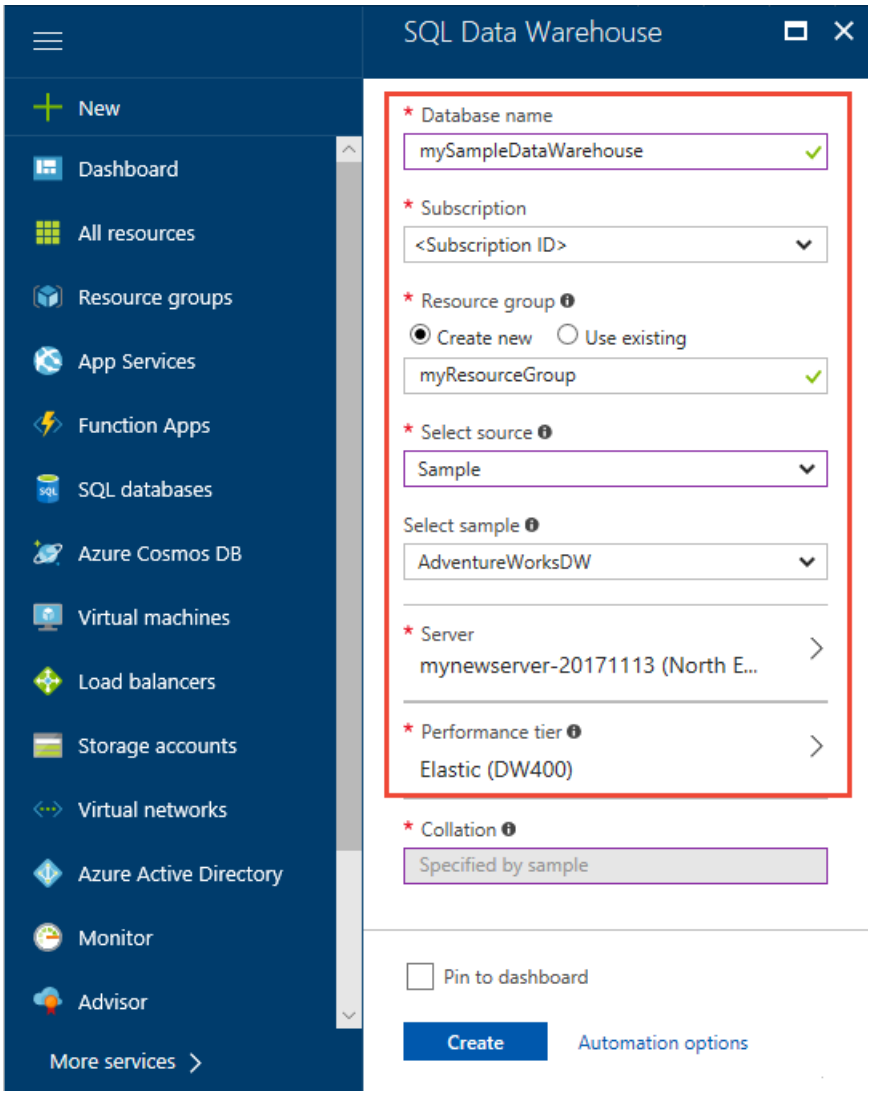
An Azure SQL data warehouse is created with a defined set of compute resources. The database is created within an Azure resource group and in an Azure SQL logical server.

Follow these steps to create a SQL data warehouse that contains the AdventureWorksDW sample data.

1. Click the **New** button in the upper left-hand corner of the Azure portal.
2. Select **Databases** from the **New** page, and select **SQL Data Warehouse** under **Featured** on the **New** page.

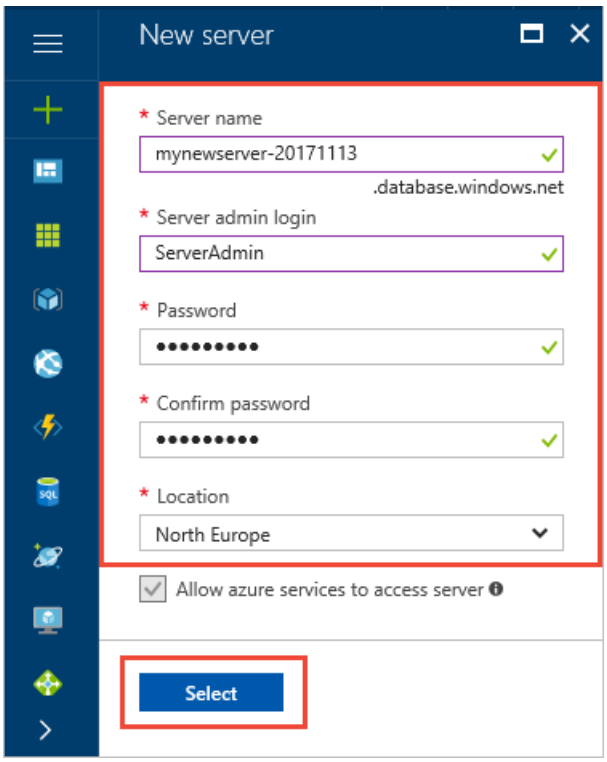


SETTING	SUGGESTED VALUE	DESCRIPTION
Database name	mySampleDataWarehouse	For valid database names, see Database Identifiers. Note, a data warehouse is a type of database.
Subscription	Your subscription	For details about your subscriptions, see Subscriptions.
Resource group	myResourceGroup	For valid resource group names, see Naming rules and restrictions.
Select source	Sample	Specifies to load a sample database. Note, a data warehouse is one type of database.
Select sample	AdventureWorksDW	Specifies to load the AdventureWorksDW sample database.

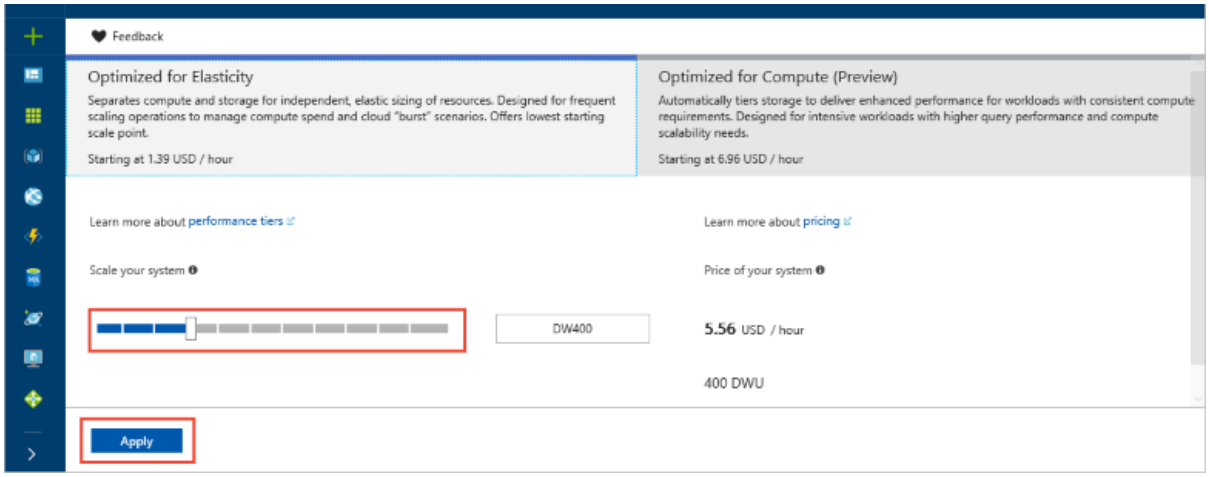


4. Click **Server** to create and configure a new server for your new database. Fill out the **New server form** with the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Server name	Any globally unique name	For valid server names, see Naming rules and restrictions.
Server admin login	Any valid name	For valid login names, see Database Identifiers.
Password	Any valid password	Your password must have at least eight characters and must contain characters from three of the following categories: upper case characters, lower case characters, numbers, and non-alphanumeric characters.
Location	Any valid location	For information about regions, see Azure Regions.

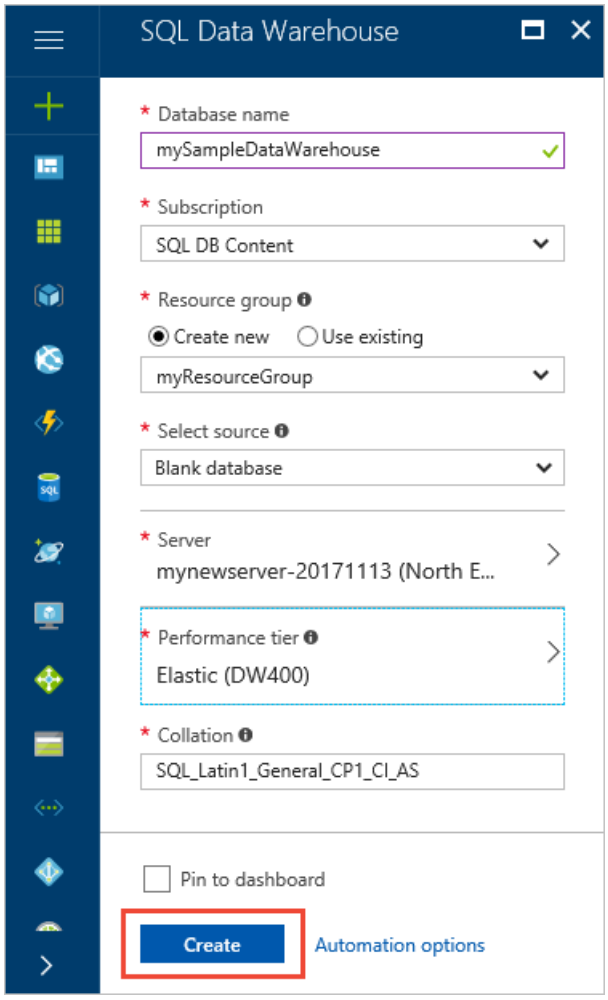


5. Click **Select**.
6. Click **Performance tier** to specify the performance configuration for the data warehouse.
7. For this tutorial, select the **Optimized for Elasticity** performance tier. The slider, by default, is set to **DW400**. Try moving it up and down to see how it works.

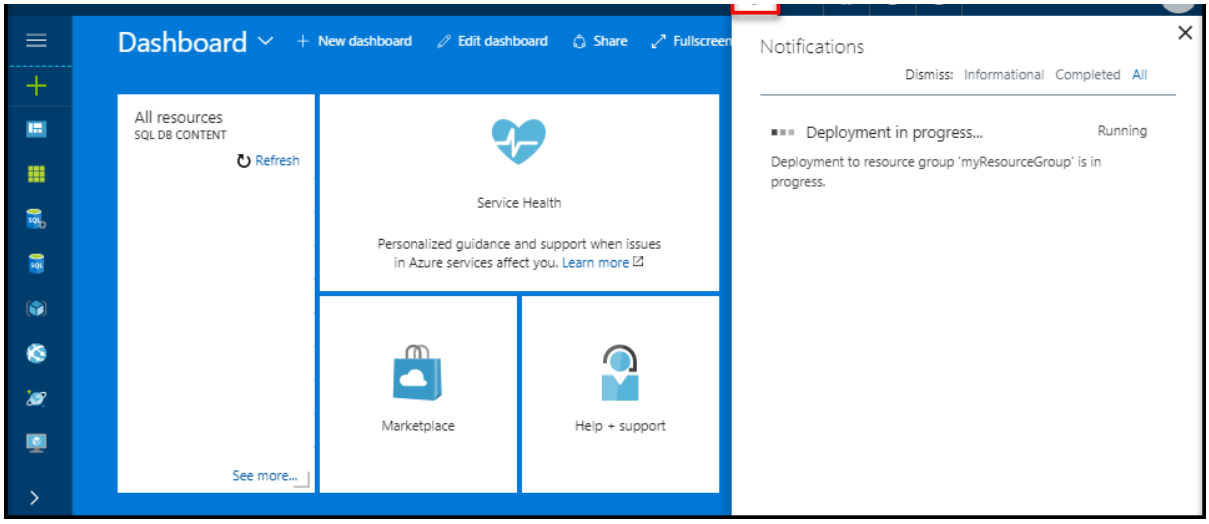


8. Click **Apply**.

9. Now that you have completed the SQL Database form, click **Create** to provision the database. Provisioning takes a few minutes.



10. On the toolbar, click **Notifications** to monitor the deployment process.



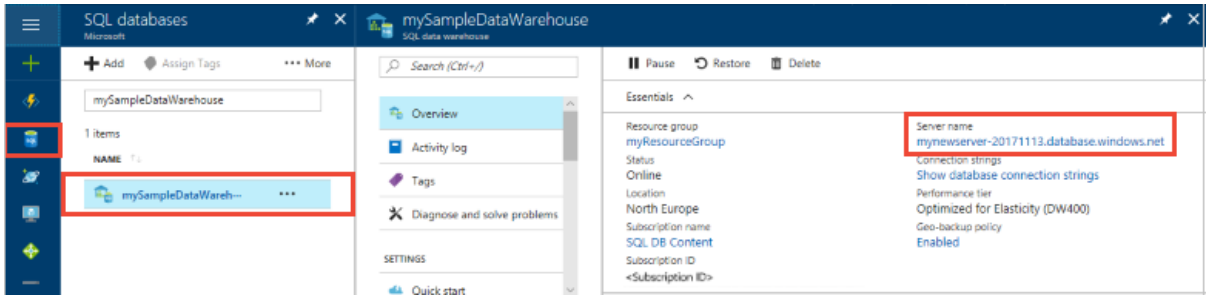
Create a server-level firewall rule

The SQL Data Warehouse service creates a firewall at the server-level that prevents external applications and tools from connecting to the server or any databases on the server. To enable connectivity, you can add firewall rules that enable connectivity for specific IP addresses. Follow these steps to create a server-level firewall rule for your client's IP address.

> **NOTE**
>
> SQL Data Warehouse communicates over port 1433. If you are trying to connect from within a corporate network, outbound traffic over port 1433 might not be allowed by your network's firewall. If so, you cannot connect to your Azure SQL Database server unless your IT department opens port 1433.

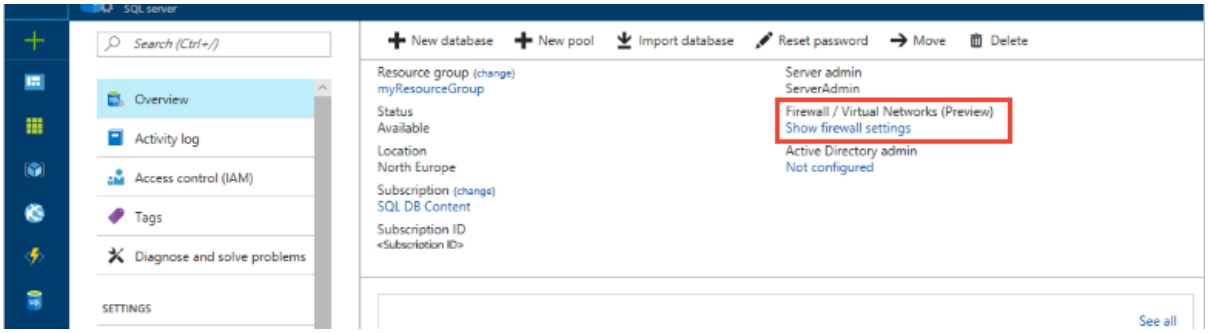
1. After the deployment completes, click **SQL databases** from the left-hand menu and then click **mySampleDatabase** on the **SQL databases** page. The overview page for your database opens, showing you the fully qualified server name (such as **mynewserver-20171113.database.windows.net**) and provides options for further configuration.

2. Copy this fully qualified server name for use to connect to your server and its databases in subsequent quick starts. To open server settings, click the server name.

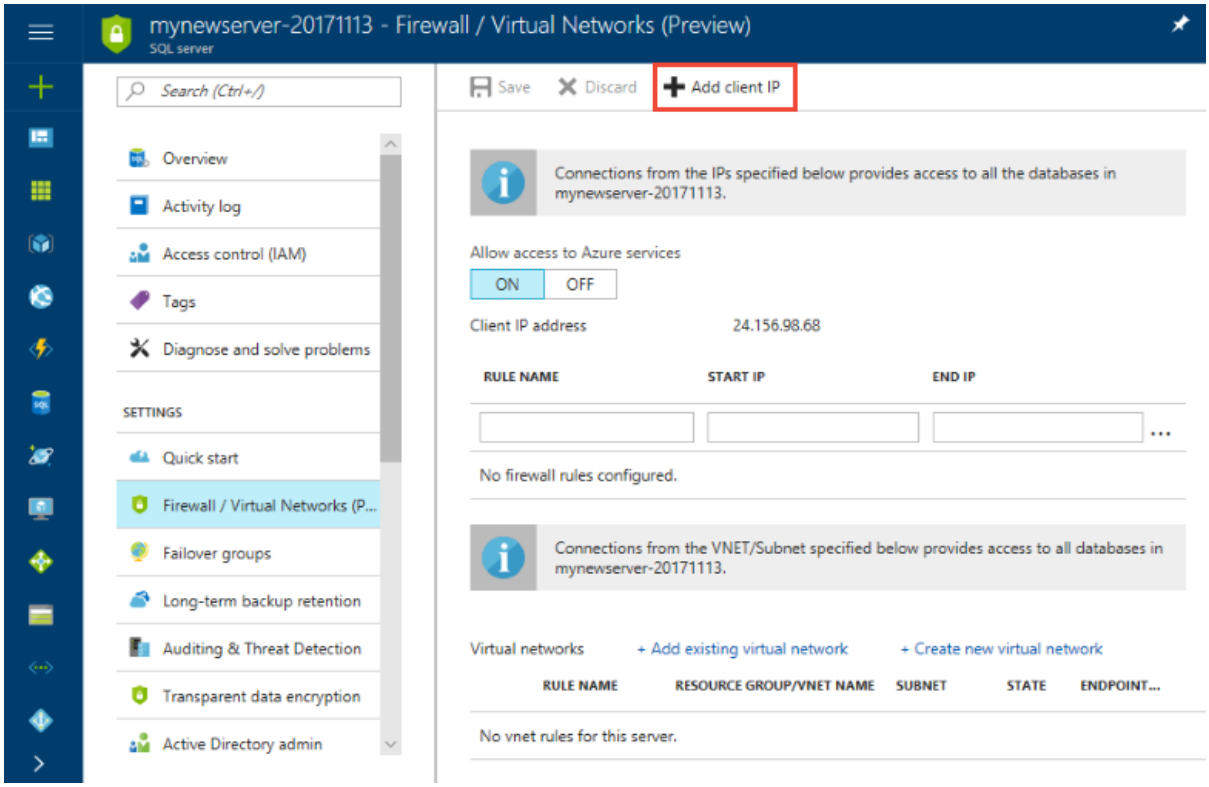


3. To open server settings,

4. click the server name.



5. Click **Show firewall settings**. The **Firewall settings** page for the SQL Database server opens.



6. To add your current IP address to a new firewall rule, click **Add client IP** on the toolbar. A firewall rule can open port 1433 for a single IP address or a range of IP addresses.
7. Click **Save**. A server-level firewall rule is created for your current IP address opening port 1433 on the logical server.
8. Click **OK** and then close the **Firewall settings** page.

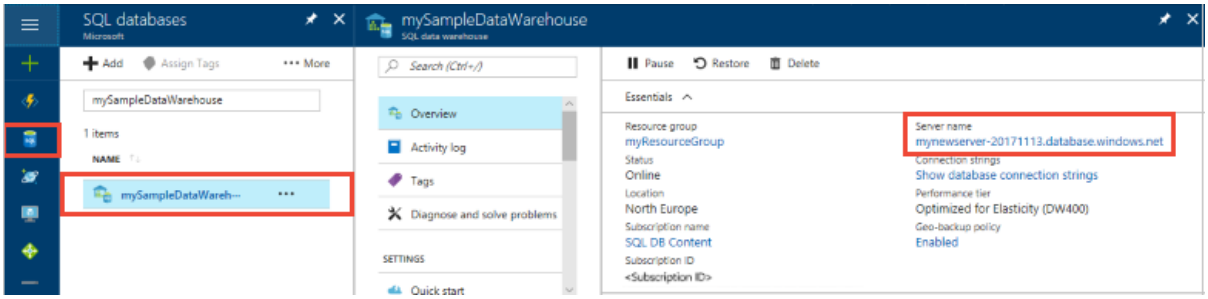
You can now connect to the SQL server and its data warehouses using this IP address. The connection works from SQL Server Management Studio or another tool of your choice. When you connect, use the ServerAdmin account you created previously.

> **IMPORTANT**
>
> By default, access through the SQL Database firewall is enabled for all Azure services. Click **OFF** on this page and then click **Save** to disable the firewall for all Azure services.

Get the fully qualified server name

Get the fully qualified server name for your SQL server in the Azure portal. Later you use the fully qualified name when connecting to the server.

2. Select **SQL Databases** from the left-hand menu, and click your database on the **SQL databases** page.
3. In the **Essentials** pane in the Azure portal page for your database, locate and then copy the **Server name**. In this example, the fully qualified name is mynewserver-20171113.database.windows.net.

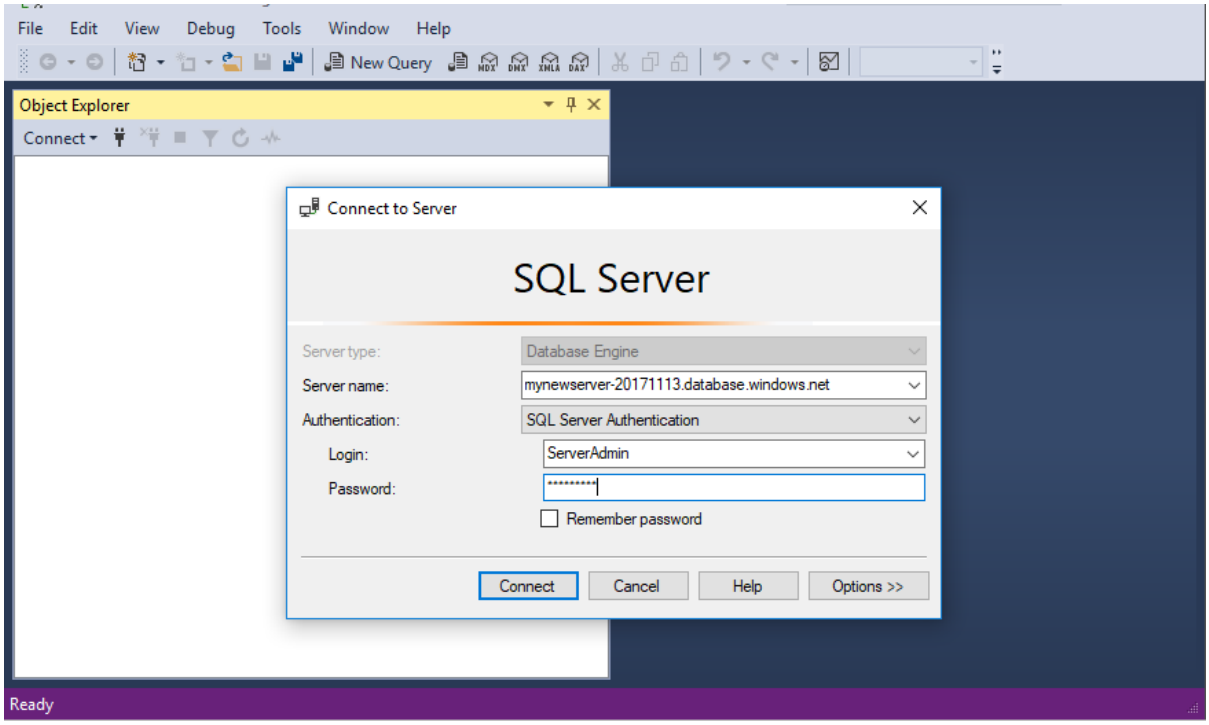


Connect to the server as server admin

This section uses SQL Server Management Studio (SSMS) to establish a connection to your Azure SQL server.

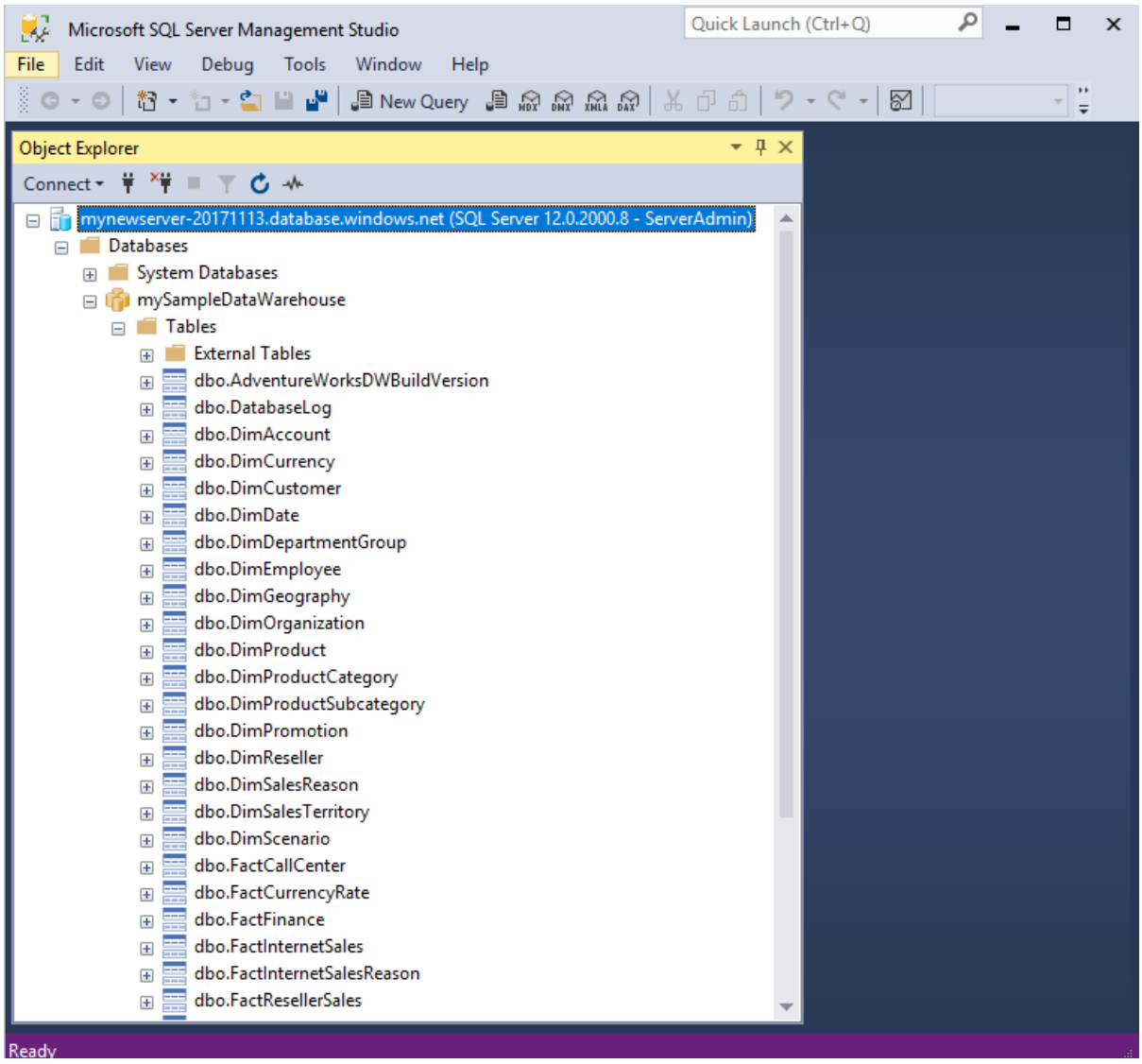
1. Open SQL Server Management Studio.
2. In the **Connect to Server** dialog box, enter the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Server type	Database engine	This value is required
Server name	The fully qualified server name	Here's an example: **mynewserver-20171113.database.windows.net**.
Authentication	SQL Server Authentication	SQL Authentication is the only authentication type that is configured in this tutorial.
Login	The server admin account	This is the account that you specified when you created the server.
Password	The password for your server admin account	This is the password that you specified when you created the server.



3. Click **Connect**. The Object Explorer window opens in SSMS.

4. In Object Explorer, expand **Databases**. Then expand **mySampleDatabase** to view the objects in your new database.



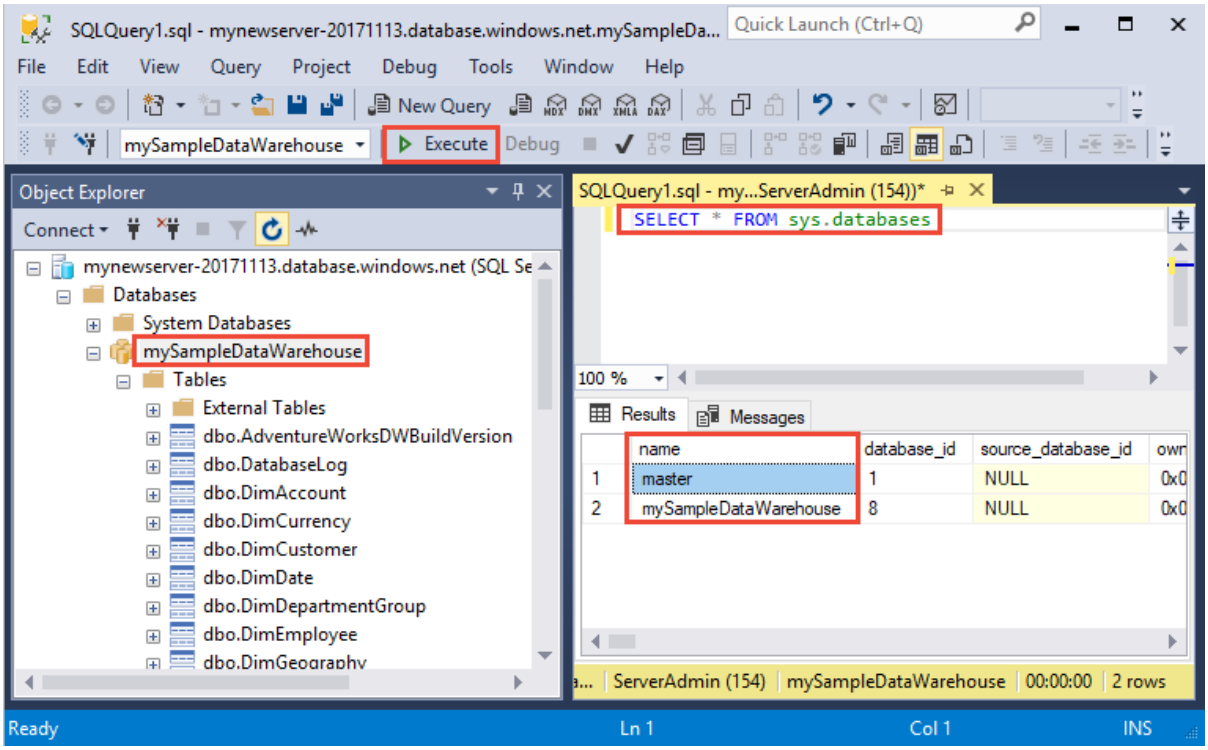
Run some queries

SQL Data Warehouse uses T-SQL as the query language. To open a query window and run some T-SQL queries, use the following steps:

1. Right-click **mySampleDataWarehouse** and select **New Query**. A new query window opens.
2. In the query window, enter the following command to see a list of databases.

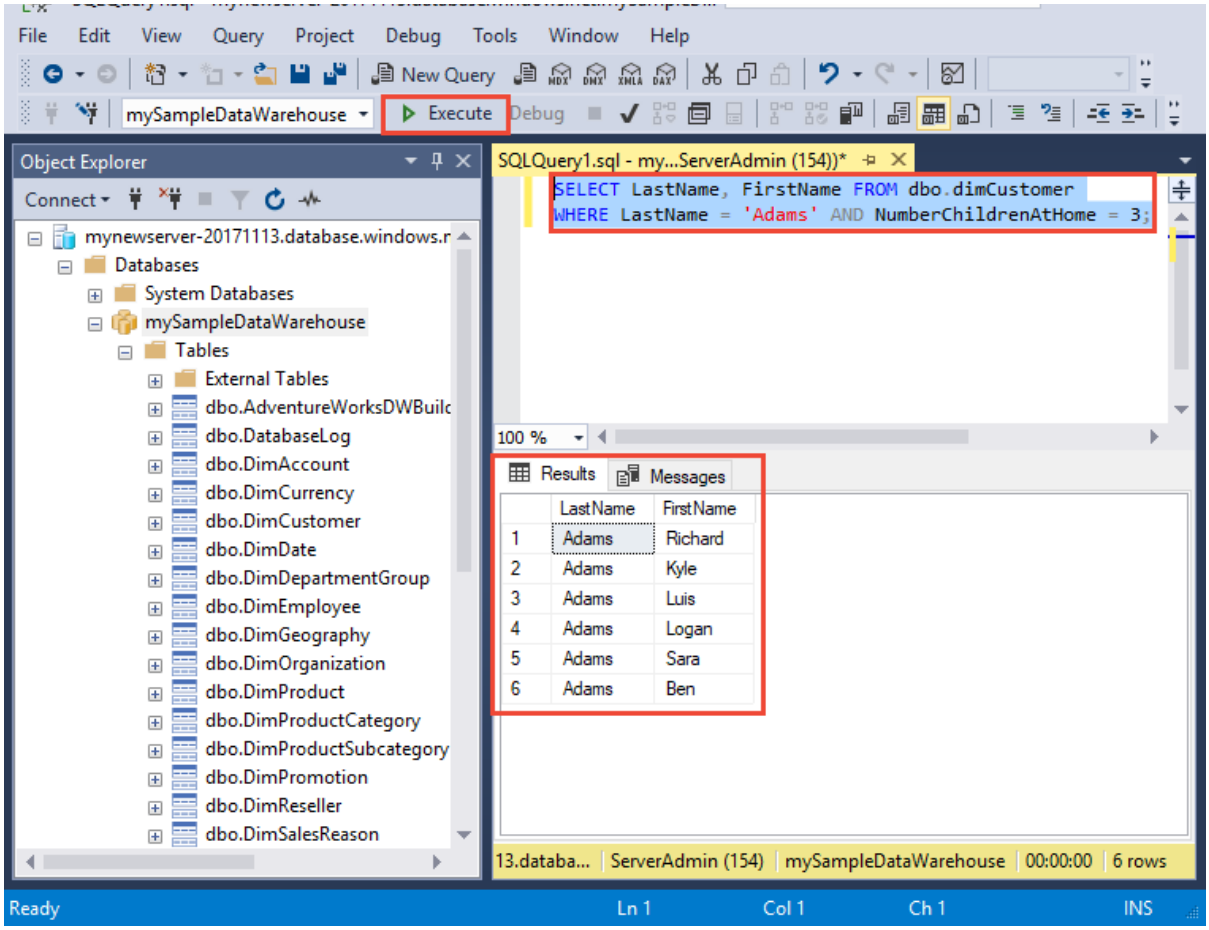
```
SELECT * FROM sys.databases
```

3. Click **Execute**. The query results show two databases: **master** and **mySampleDataWarehouse**.



4. To look at some data, use the following command to see the number of customers with last name of Adams that have three children at home. The results list six customers.

```
SELECT LastName, FirstName FROM dbo.dimCustomer
WHERE LastName = 'Adams' AND NumberChildrenAtHome = 3;
```



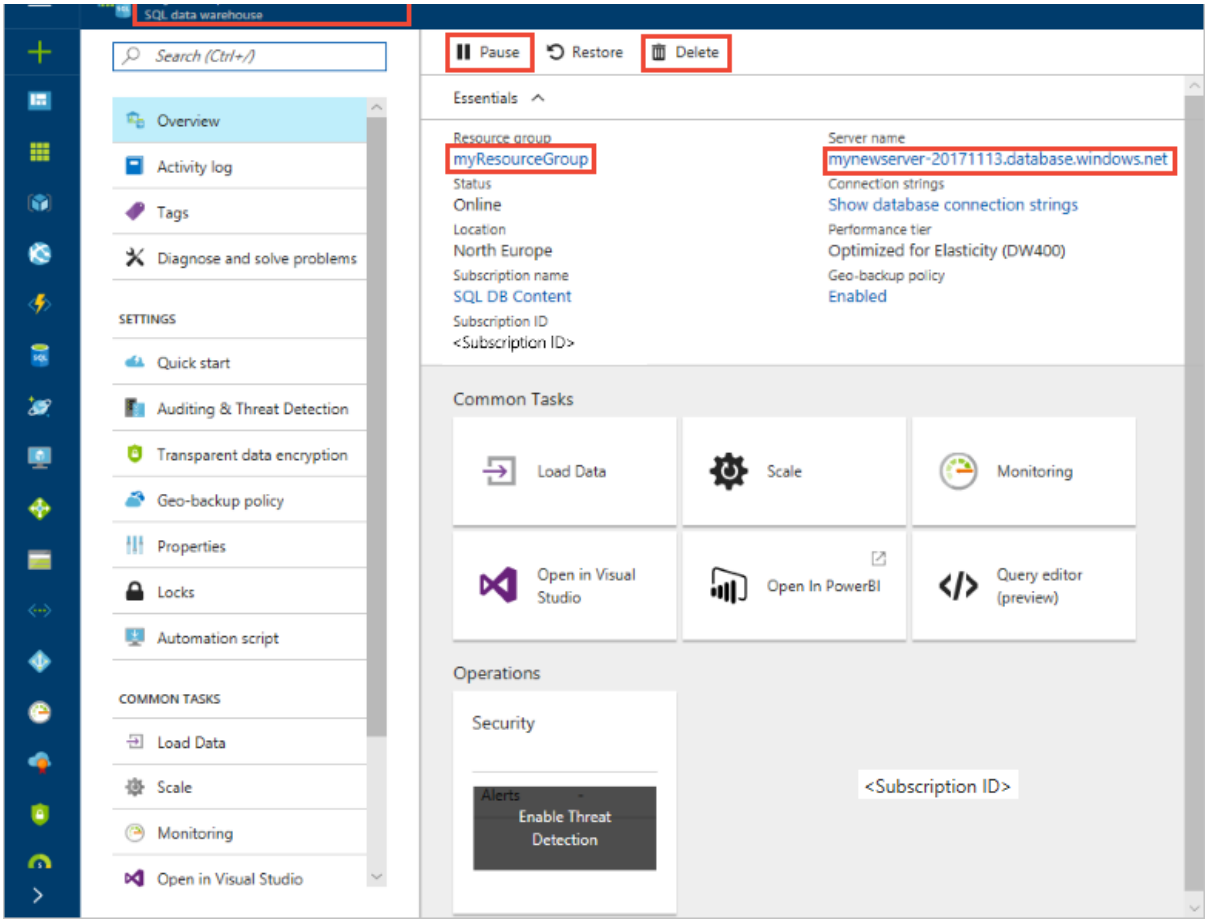
Clean up resources

You are being charged for data warehouse units and data stored your data warehouse. These compute and storage resources are billed separately.

- If you want to keep the data in storage, you can pause compute when you aren't using the data warehouse. By pausing compute, you are only charged for data storage. You can resume compute whenever you are ready to work with the data.
- If you want to remove future charges, you can delete the data warehouse.

Follow these steps to clean up resources as you desire.

1. Sign in to the Azure portal, click on your data warehouse.



2. To pause compute, click the **Pause** button. When the data warehouse is paused, you see a **Start** button. To resume compute, click **Start**.
3. To remove the data warehouse so you are not charged for compute or storage, click **Delete**.
4. To remove the SQL server you created, click **mynewserver-20171113.database.windows.net** in the previous image, and then click **Delete**. Be careful with this deletion, since deleting the server also deletes all databases assigned to the server.
5. To remove the resource group, click **myResourceGroup**, and then click **Delete resource group**.

Next steps

You have now created a data warehouse, created a firewall rule, connected to your data warehouse, and run a few queries. To learn more about Azure SQL Data Warehouse, continue to the tutorial for loading data.

Load data into a SQL data warehouse

Use PolyBase to load data from Azure blob storage to Azure SQL Data Warehouse

1/18/2018 • 16 min to read • Edit Online

PolyBase is the standard loading technology for getting data into SQL Data Warehouse. In this tutorial, you use PolyBase to load New York Taxicab data from Azure blob storage to Azure SQL Data Warehouse. The tutorial uses the Azure portal and SQL Server Management Studio (SSMS) to:

- Create a data warehouse in the Azure portal
- Set up a server-level firewall rule in the Azure portal
- Connect to the data warehouse with SSMS
- Create a user designated for loading data
- Create external tables for data in Azure blob storage
- Use the CTAS T-SQL statement to load data into your data warehouse
- View the progress of data as it is loading
- Create statistics on the newly loaded data

If you don't have an Azure subscription, create a free account before you begin.

Before you begin

Before you begin this tutorial, download and install the newest version of SQL Server Management Studio (SSMS).

Log in to the Azure portal

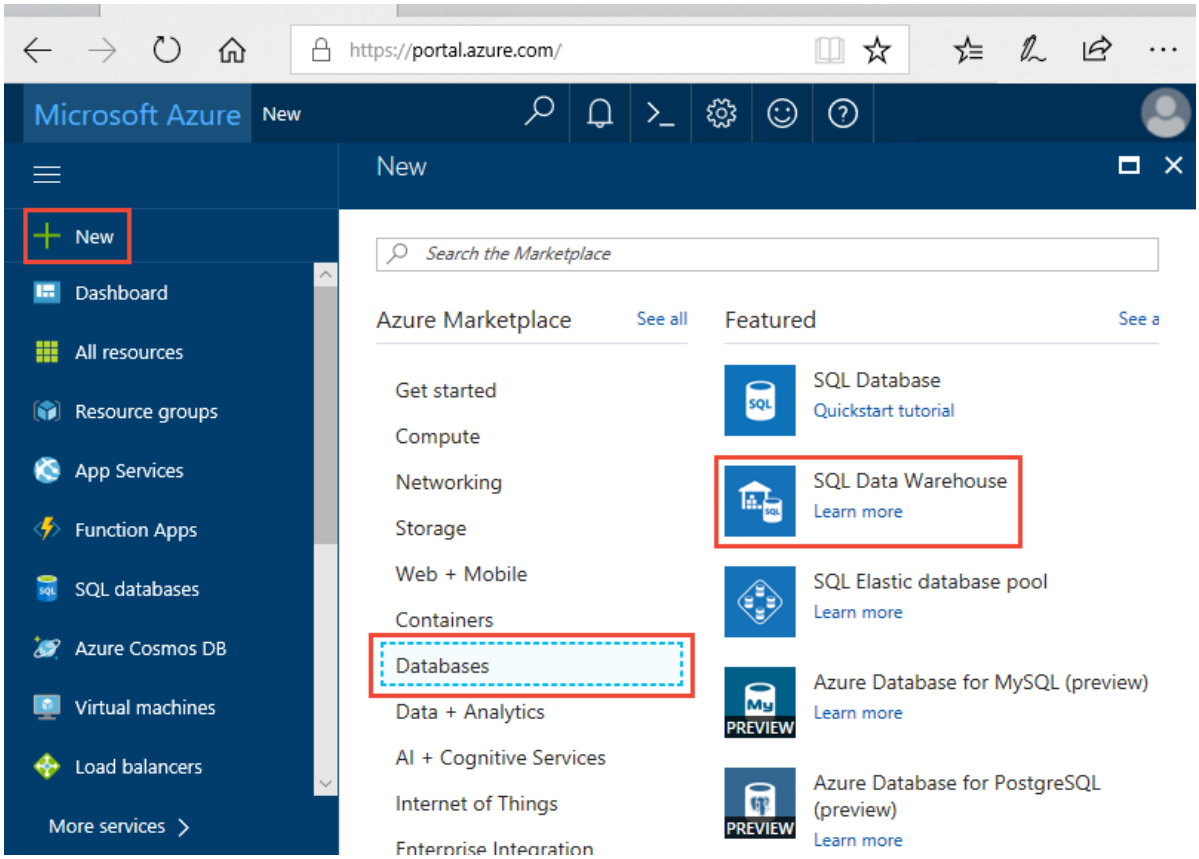
Log in to the Azure portal.

Create a blank SQL data warehouse

An Azure SQL data warehouse is created with a defined set of compute resources. The database is created within an Azure resource group and in an Azure SQL logical server.

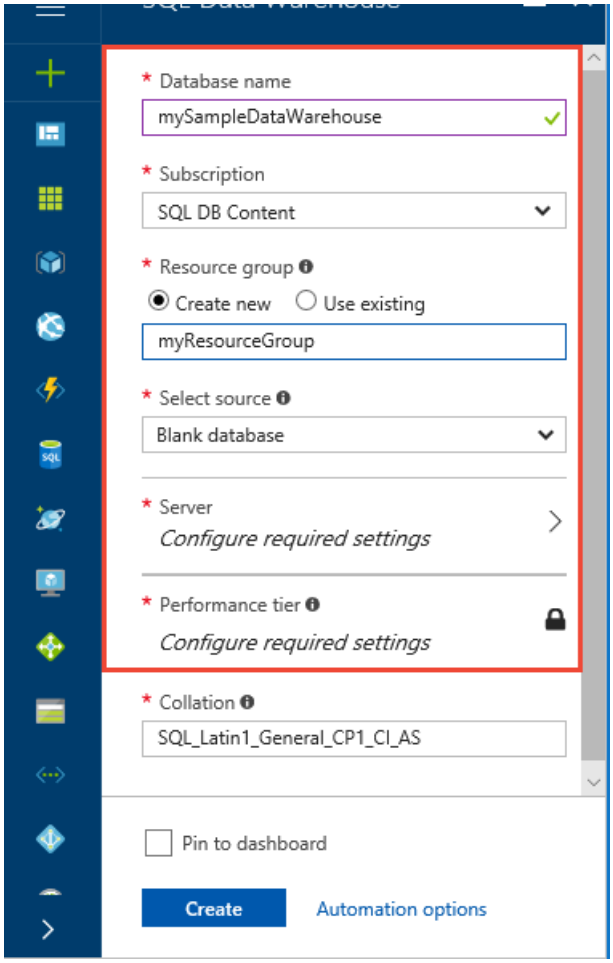
Follow these steps to create a blank SQL data warehouse.

1. Click the **New** button in the upper left-hand corner of the Azure portal.
2. Select **Databases** from the **New** page, and select **SQL Data Warehouse** under **Featured** on the **New** page.



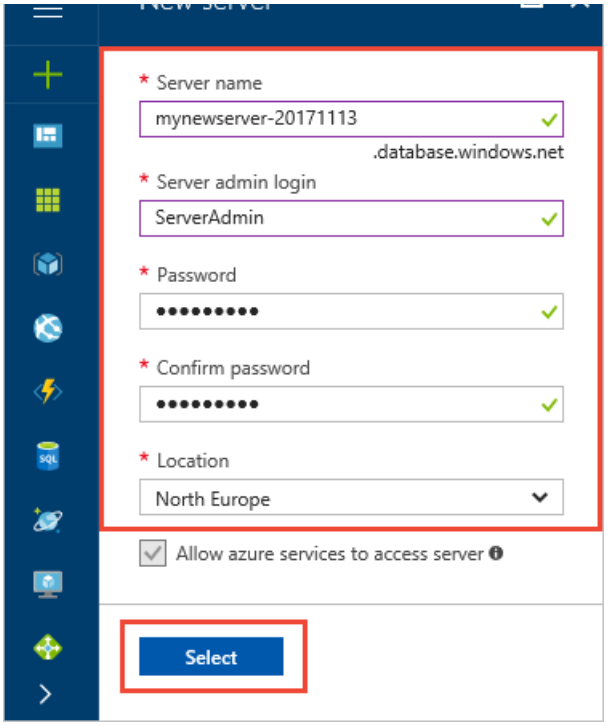
3. Fill out the SQL Data Warehouse form with the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Database name	mySampleDataWarehouse	For valid database names, see Database Identifiers.
Subscription	Your subscription	For details about your subscriptions, see Subscriptions.
Resource group	myResourceGroup	For valid resource group names, see Naming rules and restrictions.
Select source	Blank database	Specifies to create a blank database. Note, a data warehouse is one type of database.

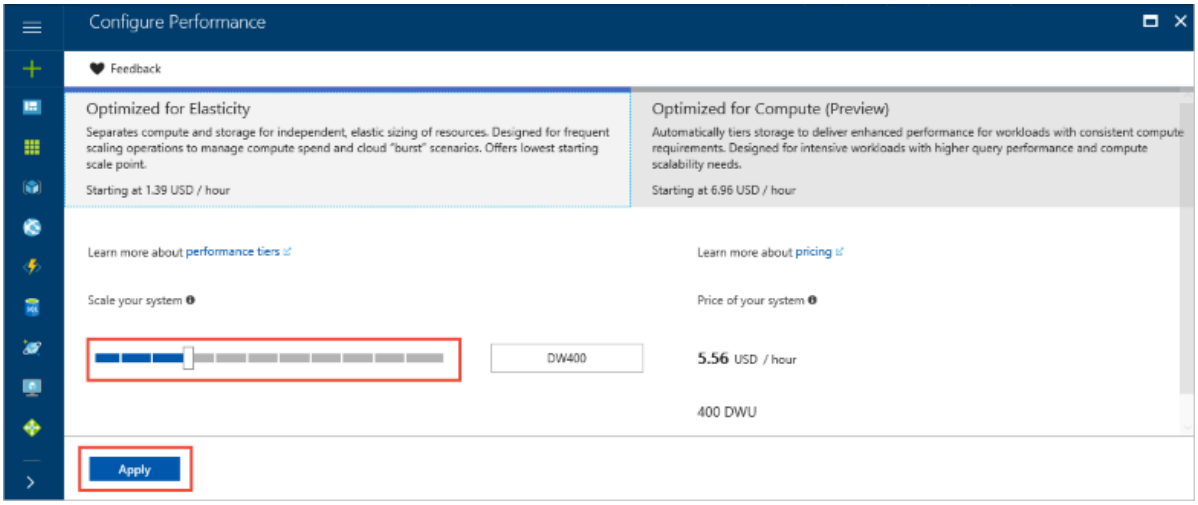


4. Click **Server** to create and configure a new server for your new database. Fill out the **New server form** with the following information:

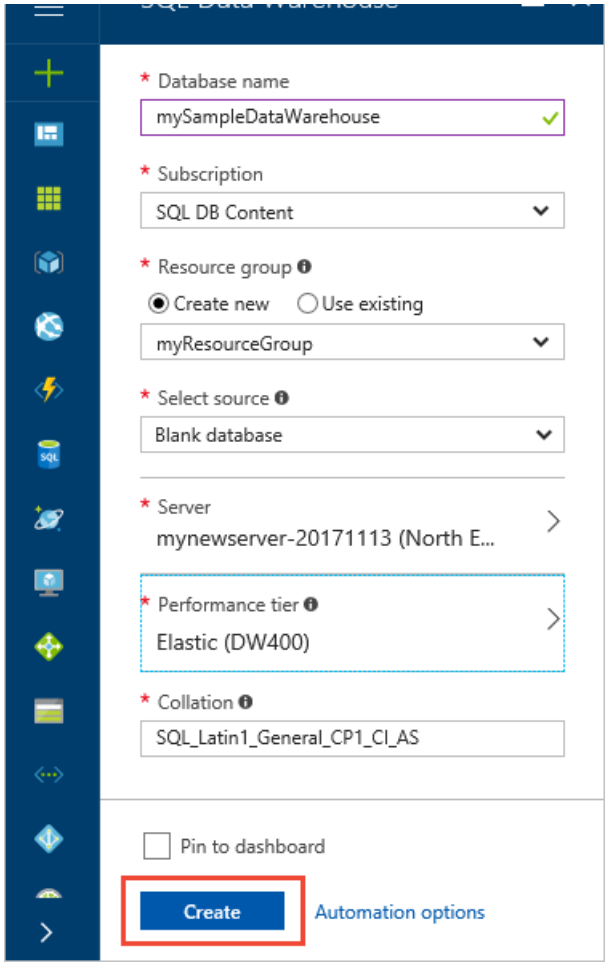
SETTING	SUGGESTED VALUE	DESCRIPTION
Server name	Any globally unique name	For valid server names, see Naming rules and restrictions.
Server admin login	Any valid name	For valid login names, see Database Identifiers.
Password	Any valid password	Your password must have at least eight characters and must contain characters from three of the following categories: upper case characters, lower case characters, numbers, and non-alphanumeric characters.
Location	Any valid location	For information about regions, see Azure Regions.



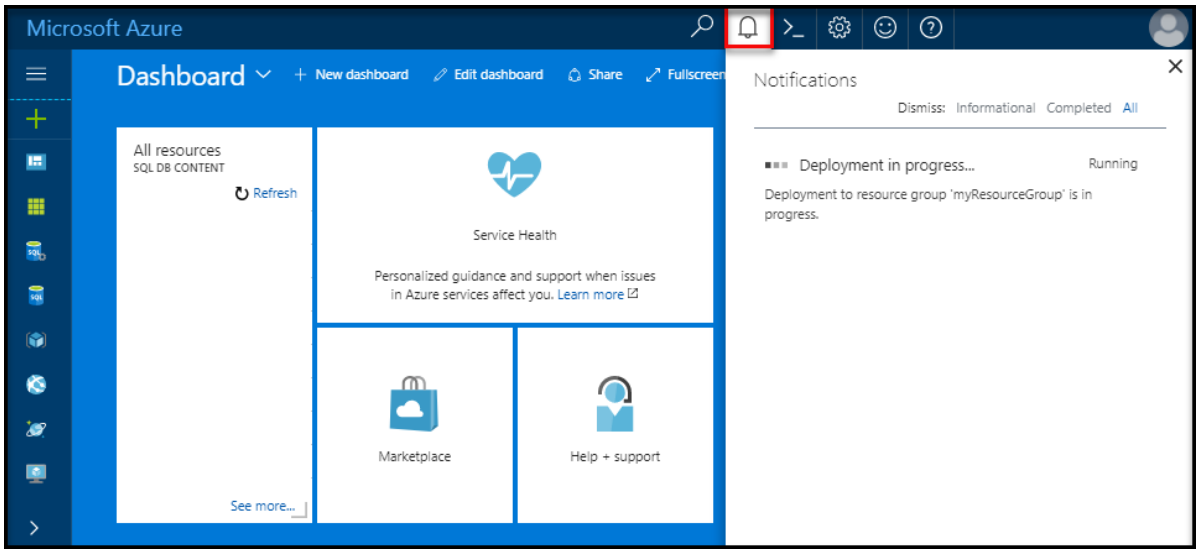
5. Click **Select**.
6. Click **Performance tier** to specify whether the data warehouse is optimized for elasticity or compute, and the number of data warehouse units.
7. For this tutorial, select the **Optimized for Elasticity** service tier. The slider, by default, is set to **DW400**. Try moving it up and down to see how it works.



8. Click **Apply**.
9. In the SQL Data Warehouse page, select a **collation** for the blank database. For this tutorial, use the default value. For more information about collations, see Collations
10. Now that you have completed the SQL Database form, click **Create** to provision the database. Provisioning takes a few minutes.



11. On the toolbar, click **Notifications** to monitor the deployment process.



Create a server-level firewall rule

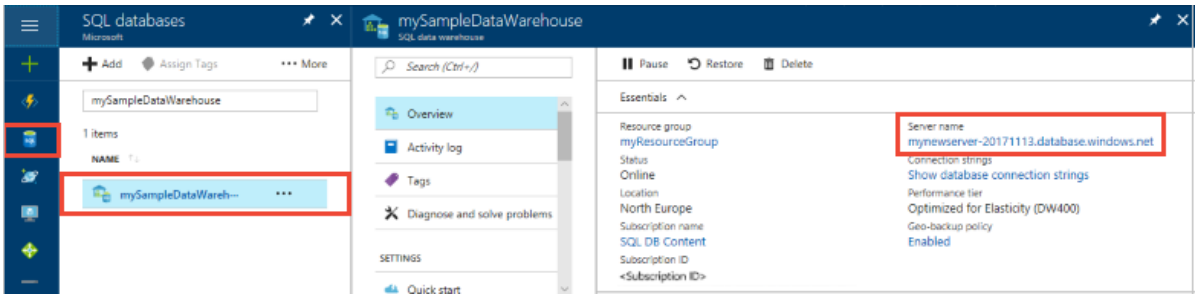
The SQL Data Warehouse service creates a firewall at the server-level that prevents external applications and tools from connecting to the server or any databases on the server. To enable connectivity, you can add firewall rules that enable connectivity for specific IP addresses. Follow these steps to create a server-level firewall rule for your client's IP address.

NOTE

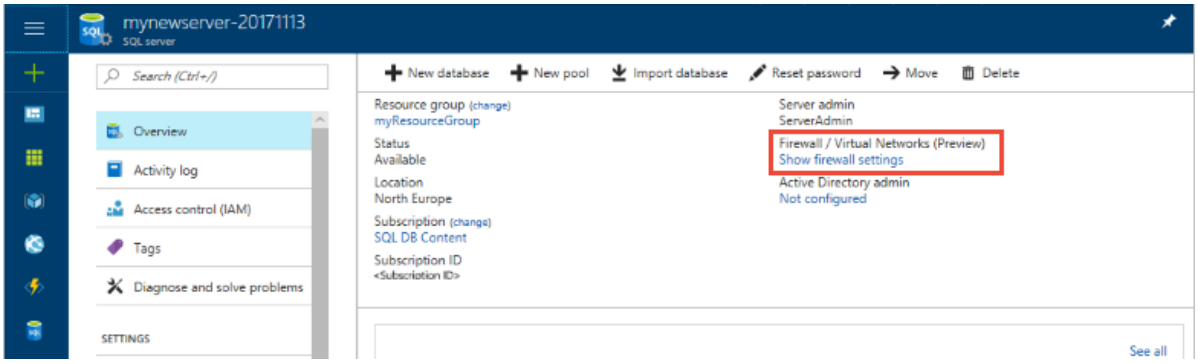
SQL Data Warehouse communicates over port 1433. If you are trying to connect from within a corporate network, outbound traffic over port 1433 might not be allowed by your network's firewall. If so, you cannot connect to your Azure SQL Database server unless your IT department opens port 1433.

1. After the deployment completes, click **SQL databases** from the left-hand menu and then click **mySampleDatabase** on the **SQL databases** page. The overview page for your database opens, showing you the fully qualified server name (such as **mynewserver-20171113.database.windows.net**) and provides options for further configuration.

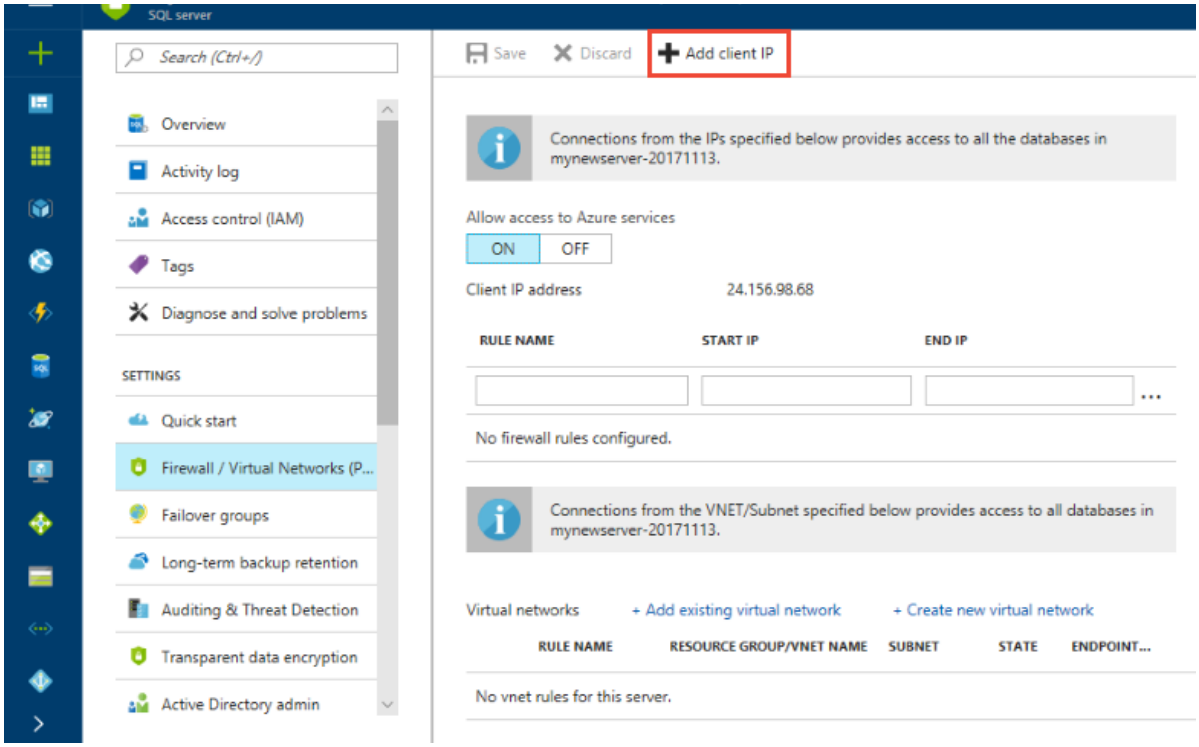
2. Copy this fully qualified server name for use to connect to your server and its databases in subsequent quick starts. Then click on the server name to open server settings.



3. Click the server name to open server settings.



4. Click **Show firewall settings**. The **Firewall settings** page for the SQL Database server opens.



5. Click **Add client IP** on the toolbar to add your current IP address to a new firewall rule. A firewall rule can open port 1433 for a single IP address or a range of IP addresses.
6. Click **Save**. A server-level firewall rule is created for your current IP address opening port 1433 on the logical server.
7. Click **OK** and then close the **Firewall settings** page.

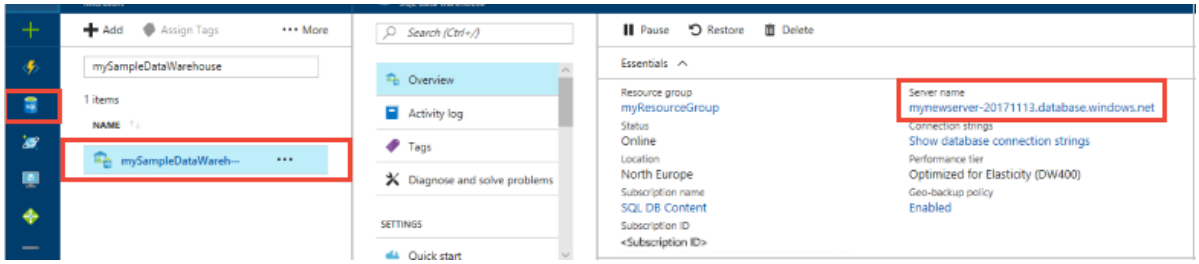
You can now connect to the SQL server and its data warehouses using this IP address. The connection works from SQL Server Management Studio or another tool of your choice. When you connect, use the ServerAdmin account you created previously.

> **IMPORTANT**
> By default, access through the SQL Database firewall is enabled for all Azure services. Click **OFF** on this page and then click **Save** to disable the firewall for all Azure services.

Get the fully qualified server name

Get the fully qualified server name for your SQL server in the Azure portal. Later you will use the fully qualified name when connecting to the server.

1. Log in to the Azure portal.
2. Select **SQL Databases** from the left-hand menu, and click your database on the **SQL databases** page.
3. In the **Essentials** pane in the Azure portal page for your database, locate and then copy the **Server name**. In this example, the fully qualified name is mynewserver-20171113.database.windows.net.

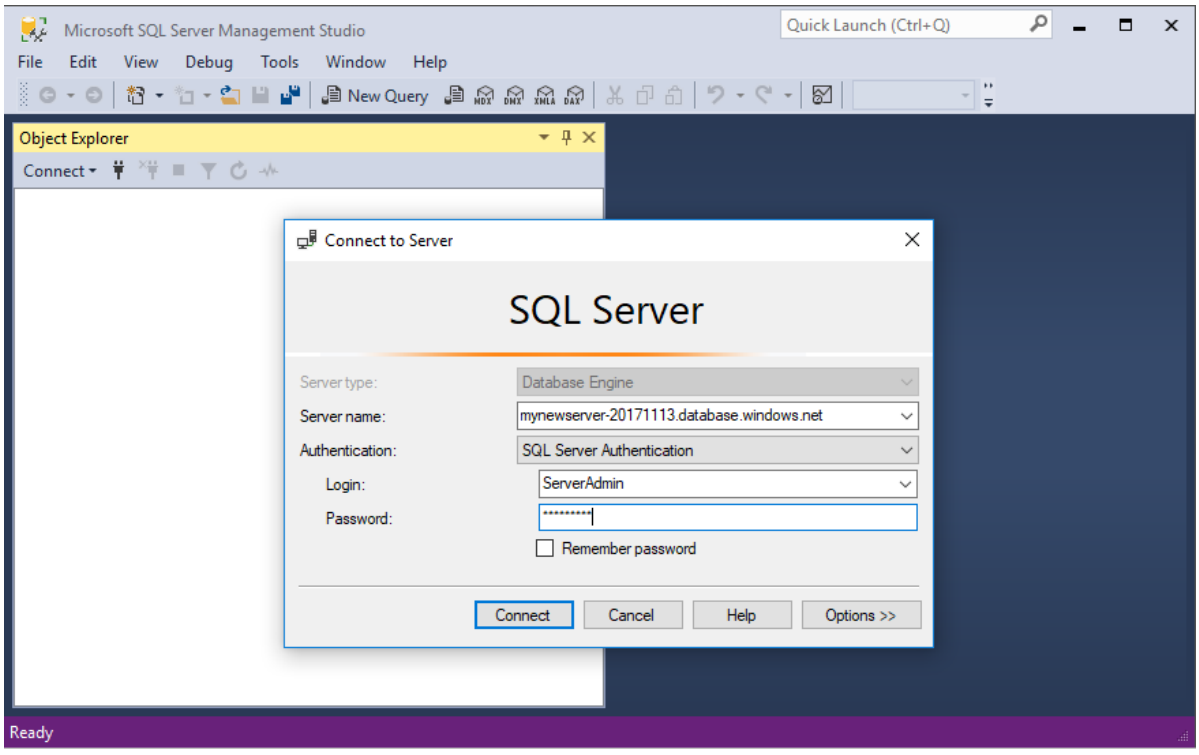


Connect to the server as server admin

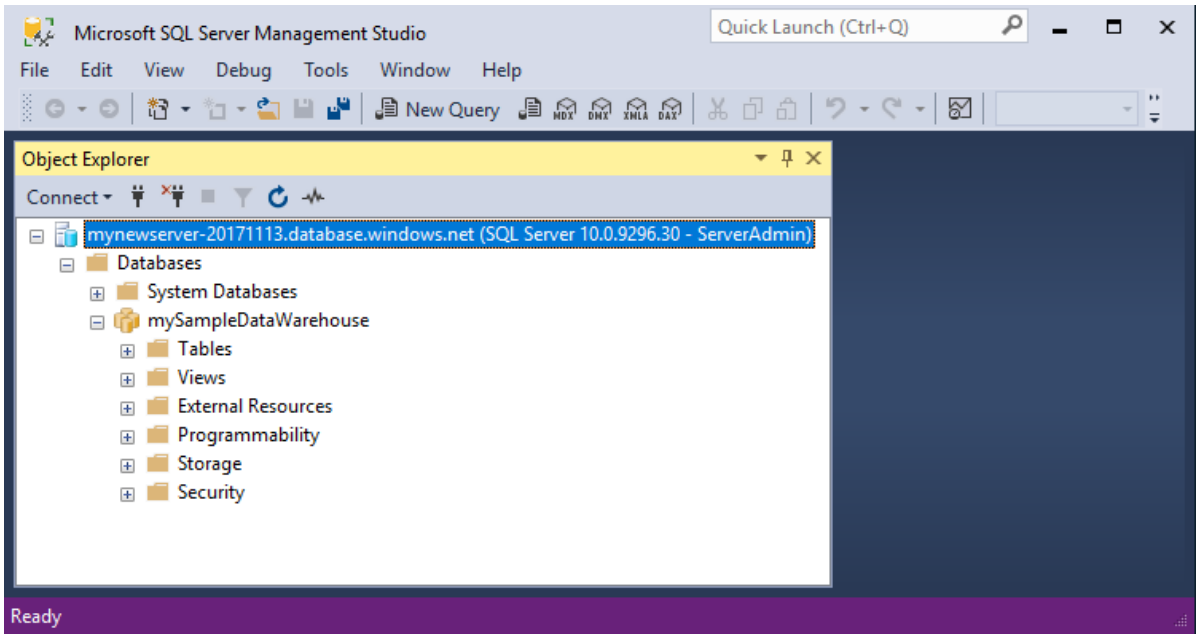
This section uses SQL Server Management Studio (SSMS) to establish a connection to your Azure SQL server.

1. Open SQL Server Management Studio.
2. In the **Connect to Server** dialog box, enter the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Server type	Database engine	This value is required
Server name	The fully qualified server name	The name should be something like this: **mynewserver-20171113.database.windows.net**.
Authentication	SQL Server Authentication	SQL Authentication is the only authentication type that we have configured in this tutorial.
Login	The server admin account	This is the account that you specified when you created the server.
Password	The password for your server admin account	This is the password that you specified when you created the server.



4. In Object Explorer, expand **Databases**. Then expand **System databases** and **master** to view the objects in the master database. Expand **mySampleDatabase** to view the objects in your new database.



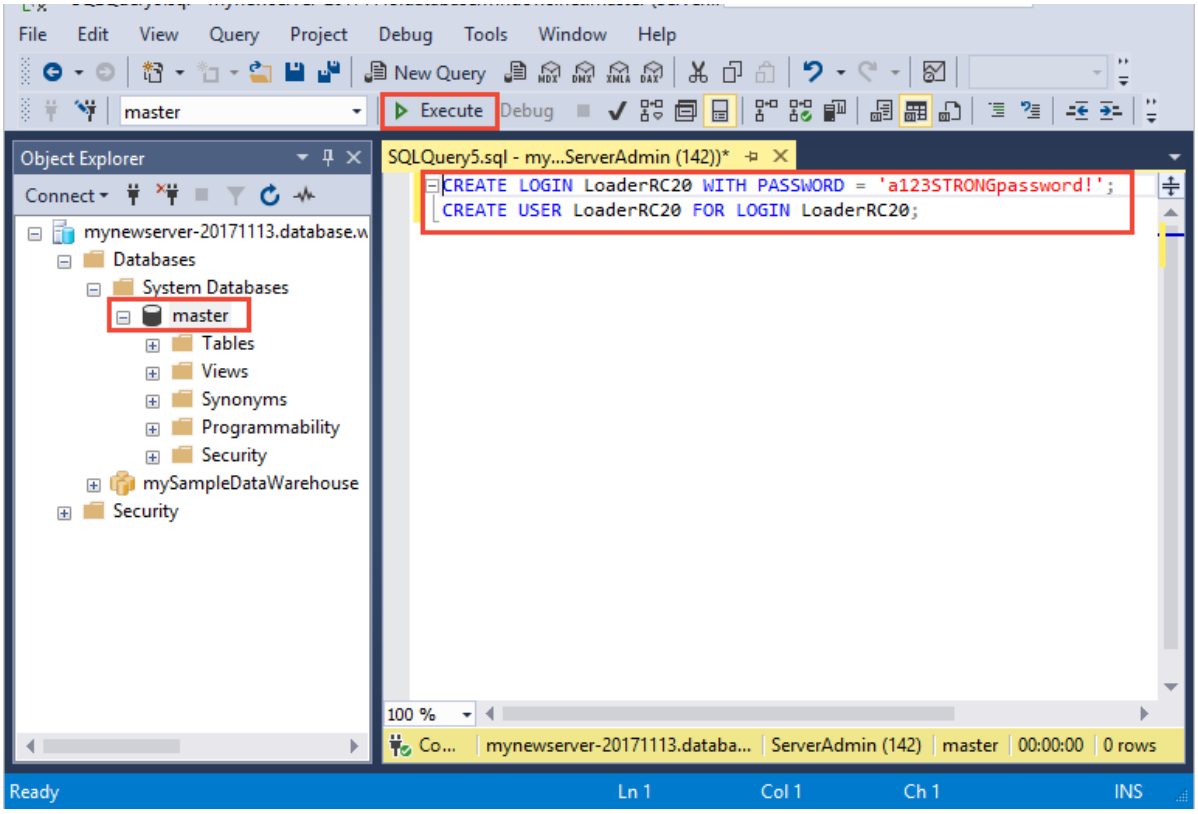
Create a user for loading data

The server admin account is meant to perform management operations, and is not suited for running queries on user data. Loading data is a memory-intensive operation. Memory maximums are defined according to performance tier, and resource class.

It's best to create a login and user that is dedicated for loading data. Then add the loading user to a resource class that enables an appropriate maximum memory allocation.

Since you are currently connected as the server admin, you can create logins and users. Use these steps to create a login and user called **LoaderRC20**. Then assign the user to the **staticrc20** resource class.

1. In SSMS, right-click **master** to show a drop-down menu, and choose **New Query**. A new query window opens.

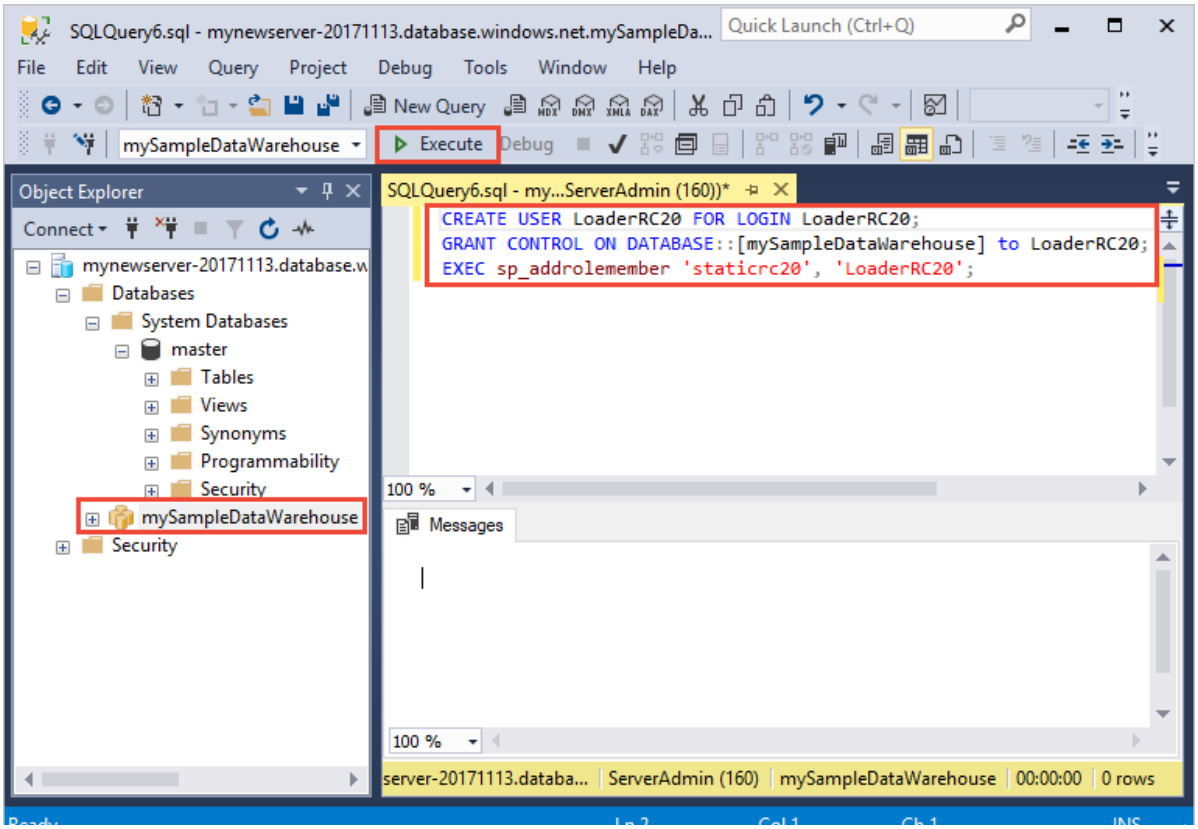


2. In the query window, enter these T-SQL commands to create a login and user named LoaderRC20, substituting your own password for 'a123STRONGpassword!'.

```
CREATE LOGIN LoaderRC20 WITH PASSWORD = 'a123STRONGpassword!';
CREATE USER LoaderRC20 FOR LOGIN LoaderRC20;
```

3. Click **Execute**.

4. Right-click **mySampleDataWarehouse**, and choose **New Query**. A new query Window opens.



login. The second line grants the new user CONTROL permissions on the new data warehouse. These permissions are similar to making the user the owner of the database. The third line adds the new user as a member of the staticrc20 resource class.

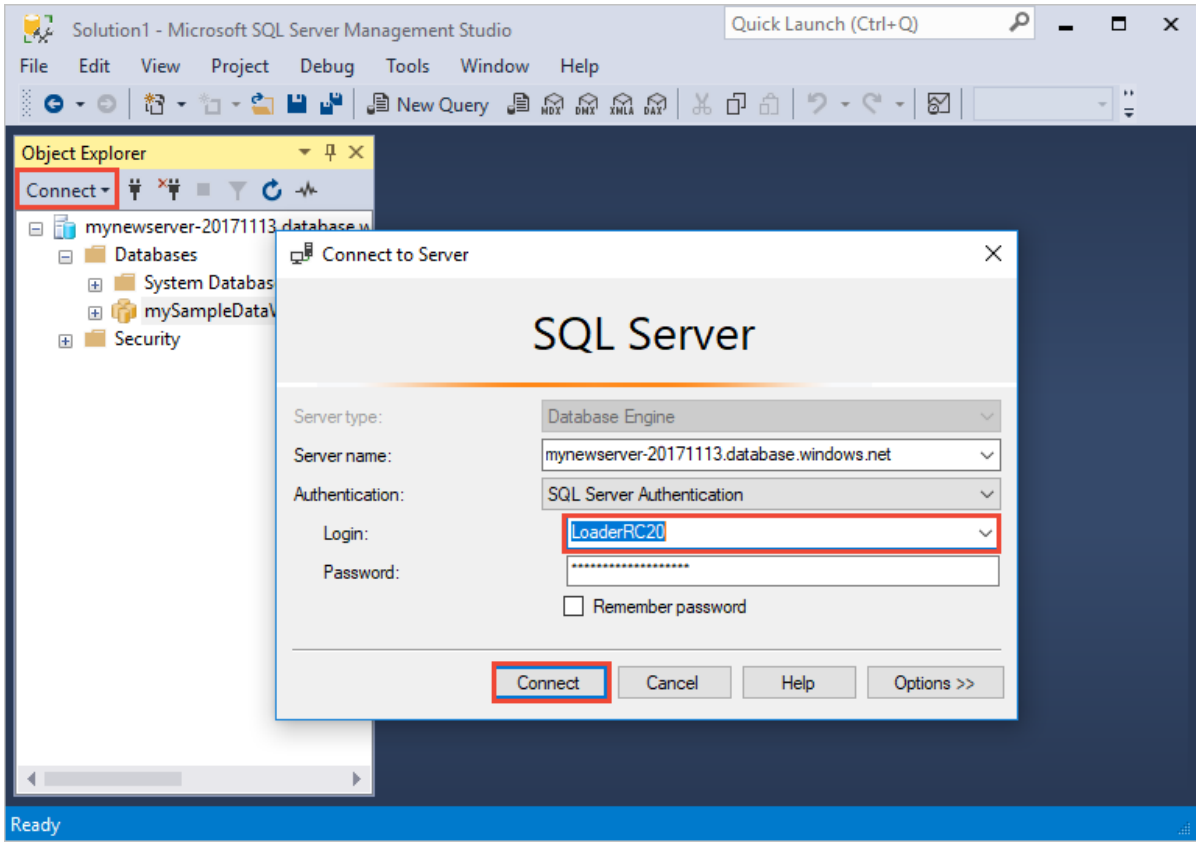
```
CREATE USER LoaderRC20 FOR LOGIN LoaderRC20;
GRANT CONTROL ON DATABASE::[mySampleDataWarehouse] to LoaderRC20;
EXEC sp_addrolemember 'staticrc20', 'LoaderRC20';
```

6. Click **Execute**.

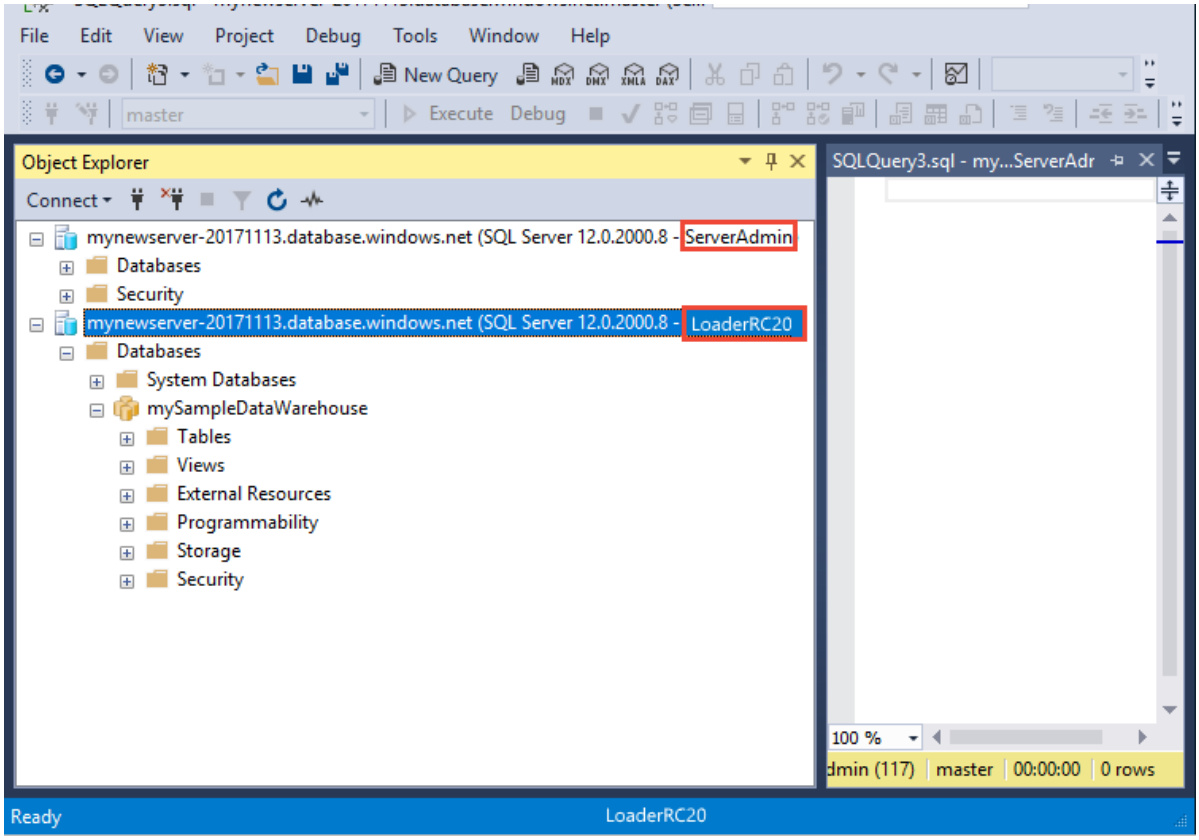
Connect to the server as the loading user

The first step toward loading data is to login as LoaderRC20.

1. In Object Explorer, click the **Connect** drop down menu and select **Database Engine**. The **Connect to Server** dialog box appears.



2. Enter the fully qualified server name, and enter **LoaderRC20** as the Login. Enter your password for LoaderRC20.
3. Click **Connect**.
4. When your connection is ready, you will see two server connections in Object Explorer. One connection as ServerAdmin and one connection as MedRCLogin.

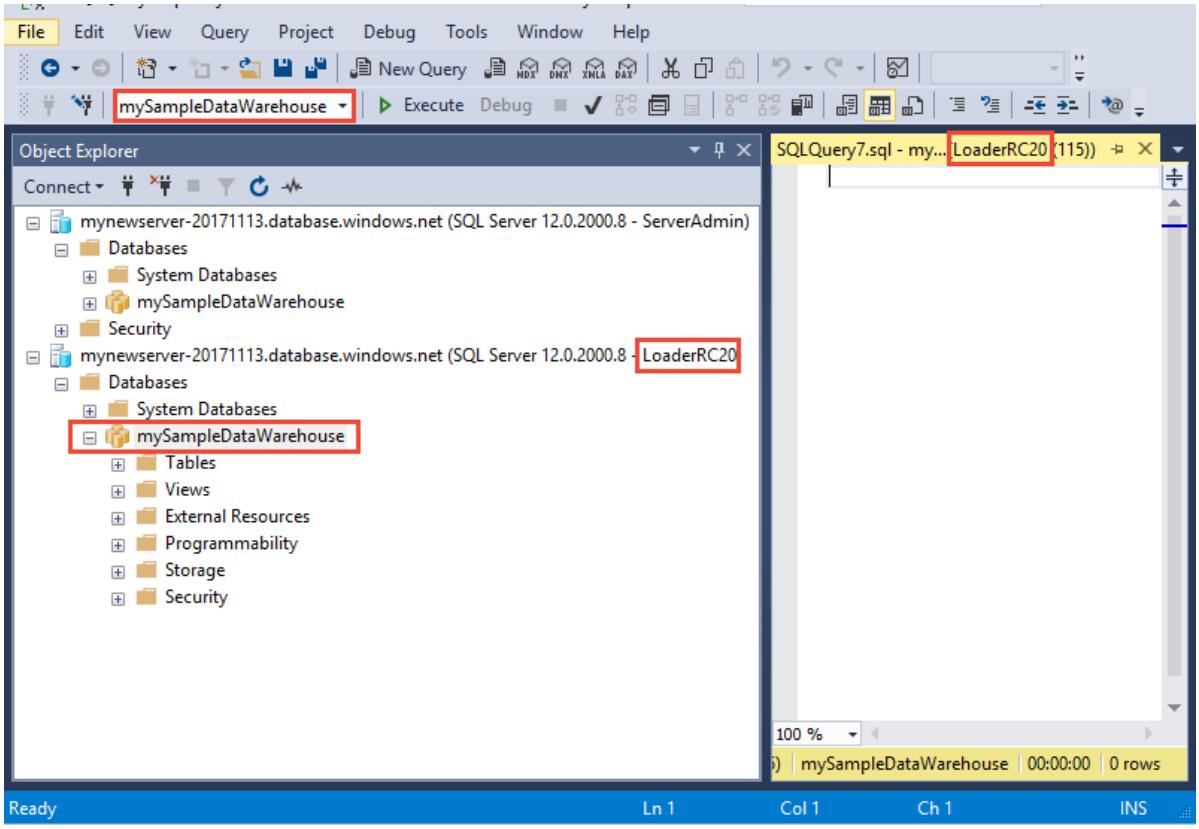


Create external tables for the sample data

You are ready to begin the process of loading data into your new data warehouse. This tutorial shows you how to use Polybase to load New York City taxi cab data from an Azure storage blob. For future reference, to learn how to get your data to Azure blob storage or to load it directly from your source into SQL Data Warehouse, see the loading overview.

Run the following SQL scripts specify information about the data you wish to load. This information includes where the data is located, the format of the contents of the data, and the table definition for the data.

1. In the previous section, you logged into your data warehouse as LoaderRC20. In SSMS, right-click your LoaderRC20 connection and select **New Query**. A new query window appears.



2. Compare your query window to the previous image. Verify your new query window is running as LoaderRC20 and performing queries on your MySampleDataWarehouse database. Use this query window to perform all of the loading steps.

3. Create a master key for the MySampleDataWarehouse database. You only need to create a master key once per database.

```
CREATE MASTER KEY;
```

4. Run the following CREATE EXTERNAL DATA SOURCE statement to define the location of the Azure blob. This is the location of the external taxi cab data. To run a command that you have appended to the query window, highlight the commands you wish to run and click **Execute**.

```
CREATE EXTERNAL DATA SOURCE NYTPublic
WITH
(
    TYPE = Hadoop,
    LOCATION = 'wasbs://2013@nytaxiblob.blob.core.windows.net/'
);
```

5. Run the following CREATE EXTERNAL FILE FORMAT T-SQL statement to specify formatting characteristics and options for the external data file. This statement specifies the external data is stored as text and the values are separated by the pipe ('|') character. The external file is compressed with Gzip.

```
CREATE EXTERNAL FILE FORMAT uncompressedcsv
WITH (
    FORMAT_TYPE = DELIMITEDTEXT,
    FORMAT_OPTIONS (
        FIELD_TERMINATOR = ',',
        STRING_DELIMITER = '',
        DATE_FORMAT = '',
        USE_TYPE_DEFAULT = False
    )
);
CREATE EXTERNAL FILE FORMAT compressedcsv
WITH (
    FORMAT_TYPE = DELIMITEDTEXT,
    FORMAT_OPTIONS ( FIELD_TERMINATOR = '|',
        STRING_DELIMITER = '',
    DATE_FORMAT = '',
        USE_TYPE_DEFAULT = False
    ),
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.GzipCodec'
);
```

6. Run the following CREATE SCHEMA statement to create a schema for your external file format. The schema provides a way to organize the external tables you are about to create.

```
CREATE SCHEMA ext;
```

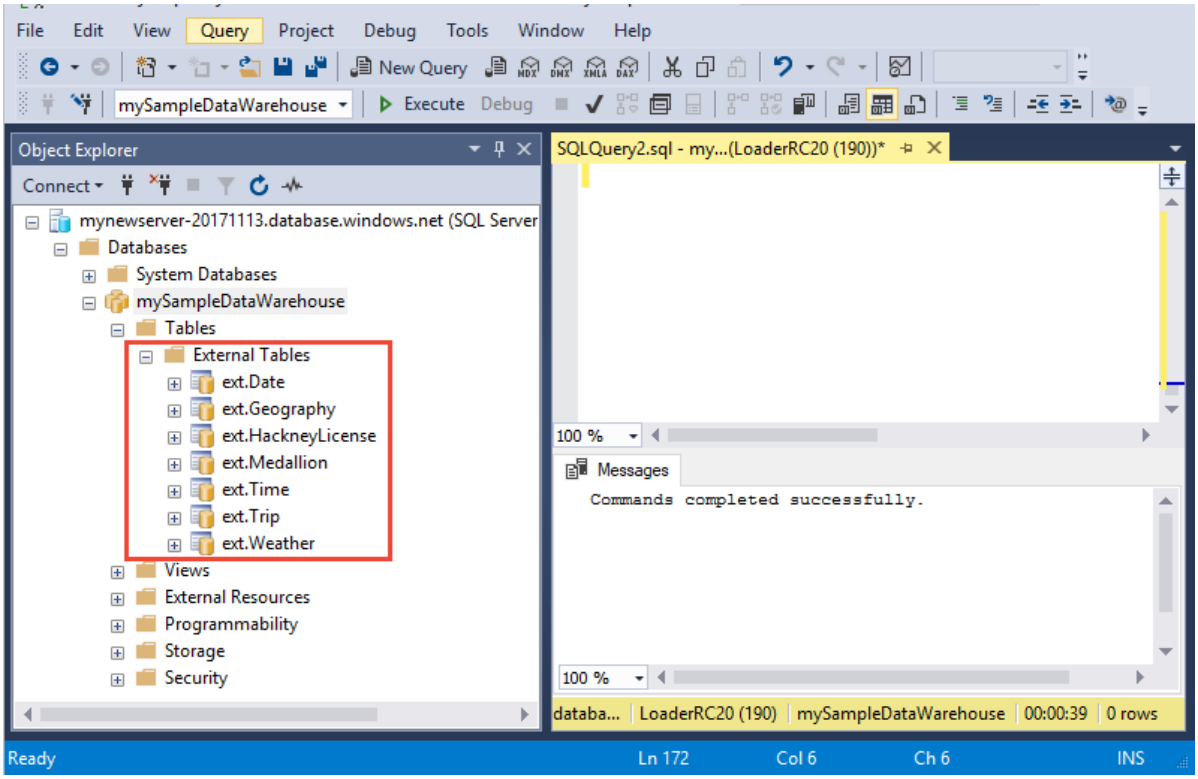
7. Create the external tables. The table definitions are stored in SQL Data Warehouse, but the tables reference data that is stored in Azure blob storage. Run the following T-SQL commands to create several external tables that all point to the Azure blob we defined previously in our external data source.

```
CREATE EXTERNAL TABLE [ext].[Date]
(
    [DateID] int NOT NULL,
    [Date] datetime NULL,
    [DateBKey] char(10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayOfMonth] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DaySuffix] varchar(4) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayName] varchar(9) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayOfWeek] char(1) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayOfWeekInMonth] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayOfWeekInYear] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayOfQuarter] varchar(3) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DayOfYear] varchar(3) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [WeekOfMonth] varchar(1) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [WeekOfQuarter] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [WeekOfYear] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [Month] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [MonthName] varchar(9) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [MonthOfQuarter] varchar(2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [Quarter] char(1) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [QuarterName] varchar(9) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [Year] char(4) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [YearName] char(7) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [MonthYear] char(10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [MMYYYY] char(6) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [FirstDayOfMonth] date NULL,
    [LastDayOfMonth] date NULL,
    [FirstDayOfQuarter] date NULL,
    [LastDayOfQuarter] date NULL,
    [FirstDayOfYear] date NULL,
    [LastDayOfYear] date NULL,
    [IsHolidayUSA] bit NULL,
```

```
)
WITH
(
    LOCATION = 'Date',
    DATA_SOURCE = NYTPublic,
    FILE_FORMAT = uncompressedcsv,
    REJECT_TYPE = value,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[Geography]
(
    [GeographyID] int NOT NULL,
    [ZipCodeBKey] varchar(10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [County] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [City] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [State] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [Country] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [ZipCode] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
)
WITH
(
    LOCATION = 'Geography',
    DATA_SOURCE = NYTPublic,
    FILE_FORMAT = uncompressedcsv,
    REJECT_TYPE = value,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[HackneyLicense]
(
    [HackneyLicenseID] int NOT NULL,
    [HackneyLicenseBKey] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [HackneyLicenseCode] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
)
WITH
(
    LOCATION = 'HackneyLicense',
    DATA_SOURCE = NYTPublic,
    FILE_FORMAT = uncompressedcsv,
    REJECT_TYPE = value,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[Medallion]
(
    [MedallionID] int NOT NULL,
    [MedallionBKey] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [MedallionCode] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
)
WITH
(
    LOCATION = 'Medallion',
    DATA_SOURCE = NYTPublic,
    FILE_FORMAT = uncompressedcsv,
    REJECT_TYPE = value,
    REJECT_VALUE = 0
)
;
CREATE EXTERNAL TABLE [ext].[Time]
(
    [TimeID] int NOT NULL,
    [TimeBKey] varchar(8) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [HourNumber] tinyint NOT NULL,
    [MinuteNumber] tinyint NOT NULL,
    [SecondNumber] tinyint NOT NULL,
    [TimeInSecond] int NOT NULL,
    [HourlyBucket] varchar(15) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [DayTimeBucketGroupKey] int NOT NULL,
    [DayTimeBucket] varchar(100) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL
```

```
(
    LOCATION = 'Time',
    DATA_SOURCE = NYTPublic,
    FILE_FORMAT = uncompressedcsv,
    REJECT_TYPE = value,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[Trip]
(
    [DateID] int NOT NULL,
    [MedallionID] int NOT NULL,
    [HackneyLicenseID] int NOT NULL,
    [PickupTimeID] int NOT NULL,
    [DropoffTimeID] int NOT NULL,
    [PickupGeographyID] int NULL,
    [DropoffGeographyID] int NULL,
    [PickupLatitude] float NULL,
    [PickupLongitude] float NULL,
    [PickupLatLong] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [DropoffLatitude] float NULL,
    [DropoffLongitude] float NULL,
    [DropoffLatLong] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [PassengerCount] int NULL,
    [TripDurationSeconds] int NULL,
    [TripDistanceMiles] float NULL,
    [PaymentType] varchar(50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [FareAmount] money NULL,
    [SurchargeAmount] money NULL,
    [TaxAmount] money NULL,
    [TipAmount] money NULL,
    [TollsAmount] money NULL,
    [TotalAmount] money NULL
)
WITH
(
    LOCATION = 'Trip2013',
    DATA_SOURCE = NYTPublic,
    FILE_FORMAT = compressedcsv,
    REJECT_TYPE = value,
    REJECT_VALUE = 0
);
CREATE EXTERNAL TABLE [ext].[Weather]
(
    [DateID] int NOT NULL,
    [GeographyID] int NOT NULL,
    [PrecipitationInches] float NOT NULL,
    [AvgTemperatureFahrenheit] float NOT NULL
)
WITH
(
    LOCATION = 'Weather',
    DATA_SOURCE = NYTPublic,
    FILE_FORMAT = uncompressedcsv,
    REJECT_TYPE = value,
    REJECT_VALUE = 0
)
;
```

8. In Object Explorer, expand mySampleDataWarehouse to see the list of external tables you just created.



Load the data into your data warehouse

This section uses the external tables you just defined to load the sample data from Azure Storage Blob to SQL Data Warehouse.

> **NOTE**
>
> This tutorial loads the data directly into the final table. In a production environment, you will usually use CREATE TABLE AS SELECT to load into a staging table. While data is in the staging table you can perform any necessary transformations. To append the data in the staging table to a production table, you can use the INSERT...SELECT statement. For more information, see Inserting data into a production table.

The script uses the CREATE TABLE AS SELECT (CTAS) T-SQL statement to load the data from Azure Storage Blob into new tables in your data warehouse. CTAS creates a new table based on the results of a select statement. The new table has the same columns and data types as the results of the select statement. When the select statement selects from an external table, SQL Data Warehouse imports the data into a relational table in the data warehouse.

1. Run the following script to load the data into new tables in your data warehouse.

```
CREATE TABLE [dbo].[Date]
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
)
AS SELECT * FROM [ext].[Date]
OPTION (LABEL = 'CTAS : Load [dbo].[Date]')
;
CREATE TABLE [dbo].[Geography]
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT * FROM [ext].[Geography]
OPTION (LABEL = 'CTAS : Load [dbo].[Geography]')
;
CREATE TABLE [dbo].[HackneyLicense]
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
)
AS SELECT * FROM [ext].[HackneyLicense]
OPTION (LABEL = 'CTAS : Load [dbo].[HackneyLicense]')
;
CREATE TABLE [dbo].[Medallion]
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
)
AS SELECT * FROM [ext].[Medallion]
OPTION (LABEL = 'CTAS : Load [dbo].[Medallion]')
;
CREATE TABLE [dbo].[Time]
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
)
AS SELECT * FROM [ext].[Time]
OPTION (LABEL = 'CTAS : Load [dbo].[Time]')
;
CREATE TABLE [dbo].[Weather]
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
)
AS SELECT * FROM [ext].[Weather]
OPTION (LABEL = 'CTAS : Load [dbo].[Weather]')
;
CREATE TABLE [dbo].[Trip]
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX
)
AS SELECT * FROM [ext].[Trip]
OPTION (LABEL = 'CTAS : Load [dbo].[Trip]')
;
```

2. View your data as it loads. You're loading several GBs of data and compressing it into highly performant

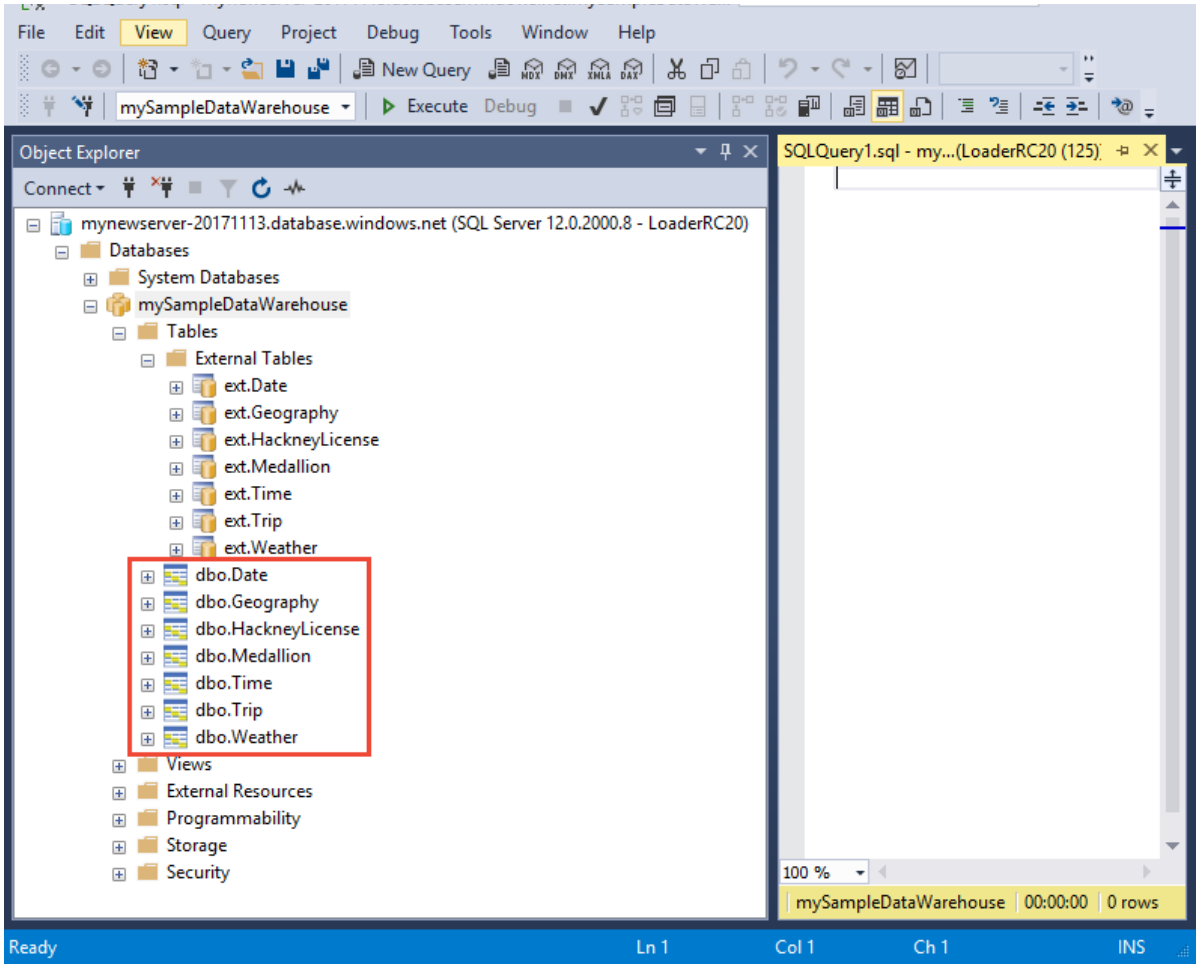
show the status of the load. After starting the query, grab a coffee and a snack while SQL Data Warehouse does some heavy lifting.

```
SELECT
    r.command,
    s.request_id,
    r.status,
    count(distinct input_name) as nbr_files,
    sum(s.bytes_processed)/1024/1024/1024 as gb_processed
FROM
    sys.dm_pdw_exec_requests r
    INNER JOIN sys.dm_pdw_dms_external_work s
    ON r.request_id = s.request_id
WHERE
    r.[label] = 'CTAS : Load [dbo].[Date]' OR
    r.[label] = 'CTAS : Load [dbo].[Geography]' OR
    r.[label] = 'CTAS : Load [dbo].[HackneyLicense]' OR
    r.[label] = 'CTAS : Load [dbo].[Medallion]' OR
    r.[label] = 'CTAS : Load [dbo].[Time]' OR
    r.[label] = 'CTAS : Load [dbo].[Weather]' OR
    r.[label] = 'CTAS : Load [dbo].[Trip]'
GROUP BY
    r.command,
    s.request_id,
    r.status
ORDER BY
    nbr_files desc,
    gb_processed desc;
```

3. View all system queries.

```
SELECT * FROM sys.dm_pdw_exec_requests;
```

4. Enjoy seeing your data nicely loaded into your data warehouse.



Create statistics on newly loaded data

SQL Data Warehouse does not auto-create or auto-update statistics. Therefore, to achieve high query performance, it's important to create statistics on each column of each table after the first load. It's also important to update statistics after substantial changes in the data.

Run these commands to create statistics on columns that are likely to be used in joins.

```sql
CREATE STATISTICS [dbo.Date DateID stats] ON dbo.Date (DateID);
CREATE STATISTICS [dbo.Trip DateID stats] ON dbo.Trip (DateID);
```

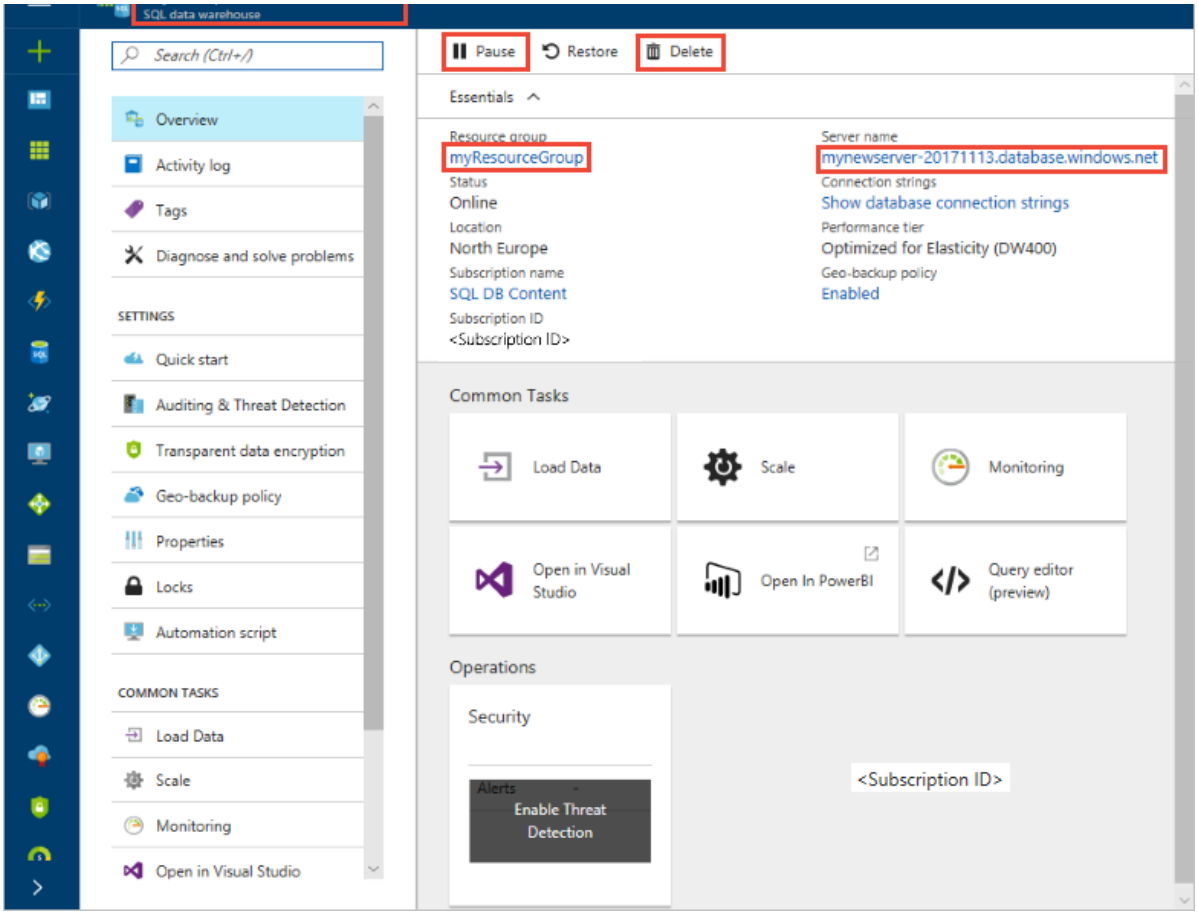
Clean up resources

You are being charged for compute resources and data that you loaded into your data warehouse. These are billed separately.

- If you want to keep the data in storage, you can pause compute when you aren't using the data warehouse. By pausing compute you will only be charge for data storage and you can resume the compute whenever you are ready to work with the data.
- If you want to remove future charges, you can delete the data warehouse.

Follow these steps to clean up resources as you desire.

1. Log in to the Azure portal, click on your data warehouse.



2. To pause compute, click the **Pause** button. When the data warehouse is paused, you will see a **Start** button. To resume compute, click **Start**.
3. To remove the data warehouse so you won't be charged for compute or storage, click **Delete**.
4. To remove the SQL server you created, click **mynewserver-20171113.database.windows.net** in the previous image, and then click **Delete**. Be careful with this as deleting the server will delete all databases assigned to the server.
5. To remove the resource group, click **myResourceGroup**, and then click **Delete resource group**.

Next steps

In this tutorial, you learned how to create a data warehouse and create a user for loading data. You created external tables to define the structure for data stored in Azure Storage Blob, and then used the PolyBase CREATE TABLE AS SELECT statement to load data into your data warehouse.

You did these things:

- Created a data warehouse in the Azure portal
- Set up a server-level firewall rule in the Azure portal
- Connected to the data warehouse with SSMS
- Created a user designated for loading data
- Created external tables for data in Azure Storage Blob
- Used the CTAS T-SQL statement to load data into your data warehouse
- Viewed the progress of data as it is loading
- Created statistics on the newly loaded data

Advance to the migration overview to learn how to migrate an existing database to SQL Data Warehouse.

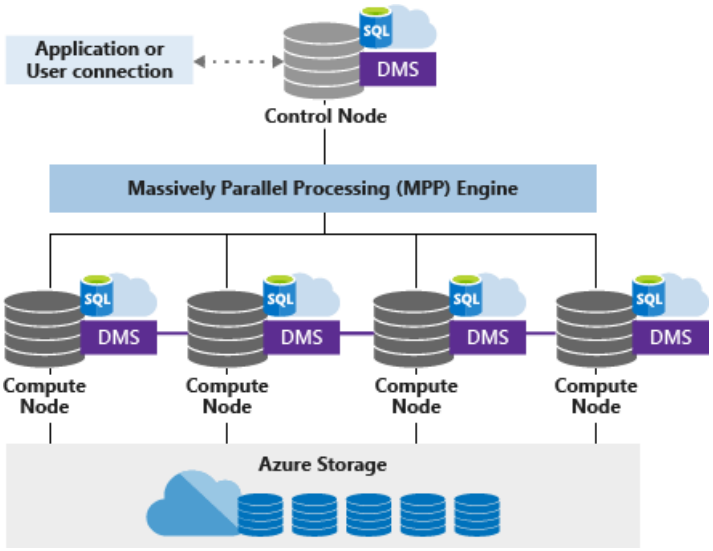
Azure SQL Data Warehouse - Massively parallel processing (MPP) architecture

11/15/2017 • 5 min to read • Edit Online

Learn how Azure SQL Data Warehouse combines massively parallel processing (MPP) with Azure storage to achieve high performance and scalability.

MPP Architecture components

SQL Data Warehouse 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. SQL Data Warehouse separates compute from storage which enables you as the user to scale compute independently of the data in your system.



SQL Data Warehouse uses a node based architecture. Applications connect and issue T-SQL commands to a Control node, which is the single point of entry for the data warehouse. 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, SQL Data Warehouse can:

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

Azure storage

SQL Data Warehouse uses Azure storage to keep your user data safe. Since your data is stored and managed by Azure storage, SQL Data Warehouse charges separately for your storage consumption. The data itself 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. SQL Data Warehouse supports these sharding patterns:

- Hash

- Replicate

Control node

The Control node is the brain of the data warehouse. 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 to SQL Data Warehouse, the Control node transforms it into queries that run against each distribution in parallel.

Compute nodes

The Compute nodes provide the computational power. Distributions map to Compute nodes for processing. As you pay for more compute resources, SQL Data Warehouse re-maps the distributions to the available Compute nodes. The number of compute nodes ranges from 1 to 60, and is determined by the service level for the data warehouse.

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.

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.

Distributions

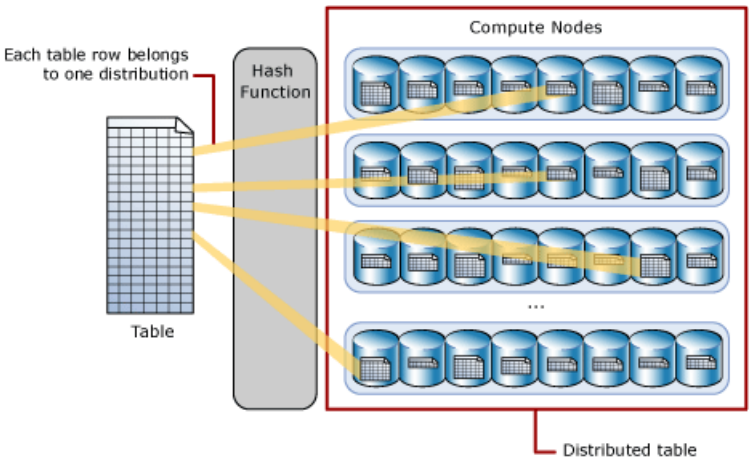
A distribution is the basic unit of storage and processing for parallel queries that run on distributed data. When SQL Data Warehouse runs a query, the work is divided into 60 smaller queries that run in parallel. Each of the 60 smaller queries runs on one of the data distributions. Each Compute node manages one or more of the 60 distributions. A data warehouse with maximum compute resources has one distribution per Compute node. A data warehouse with minimum compute resources has all the distributions on one compute node.

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, SQL Data Warehouse uses a hash function 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.

- 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.

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.

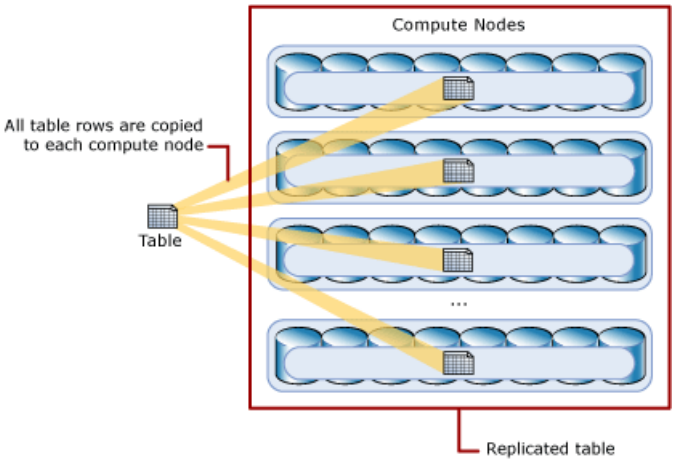
A round-robin distributed table distributes data evenly across the table but without any further optimization. A distribution is first chosen at random and then buffers of rows are assigned to distributions sequentially. It is quick to load data into a round-robin table, but query performance can often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

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 are additional overheads that are incurred when writing data which make large tables impractical.

The following diagram shows a replicated table. For SQL Data Warehouse, the replicated table is cached on the first distribution on each compute node.



Next steps

Now that you know a bit about SQL Data Warehouse, learn how to quickly create a SQL Data Warehouse and load sample data. If you are new to Azure, you may find the Azure glossary helpful as you encounter new terminology. Or look at some of these other SQL Data Warehouse Resources.

- Customer success stories
- Blogs
- Feature requests
- Videos
- Customer Advisory Team blogs
- Create support ticket
- MSDN forum

- Twitter

Azure SQL Data Warehouse performance tiers (Preview)

1/2/2018 • 7 min to read • Edit Online

SQL Data Warehouse offers two performance tiers that are optimized for analytical workloads. This article explains the concepts of performance tiers to help you choose the most suitable performance tier for your workload.

What is a performance tier?

A performance tier is an option that determines the configuration of your data warehouse. This option is one of the first choices you make when creating a data warehouse.

- The **Optimized for Elasticity performance tier** separates the compute and storage layers in the architecture. This option excels on workloads that can take full advantage of the separation between compute and storage by scaling frequently to support short periods of peak activity. This compute tier has the lowest entry price point and scales to support the majority of customer workloads.
- The **Optimized for Compute performance tier** uses the latest Azure hardware to introduce a new NVMe Solid State Disk cache that keeps the most frequently accessed data close to the CPUs, which is exactly where you want it. By automatically tiering the storage, this performance tier excels with complex queries since all IO is kept local to the compute layer. Furthermore, the columnstore is enhanced to store an unlimited amount of data in your SQL Data Warehouse. The Optimized for Compute performance tier provides the greatest level of scalability, enabling you to scale up to 30,000 compute Data Warehouse Units (cDWU). Choose this tier for workloads that requires continuous, blazing fast, performance.

Service levels

The Service Level Objective (SLO) is the scalability setting that determines the cost and performance level of your data warehouse. The service levels for the Optimized for Compute performance tier scale are measured in compute data warehouse units (cDWU), for example DW2000c. The Optimized for Elasticity service levels are measured in DWUs, for example DW2000. For more information, see What is a data warehouse unit?

In T-SQL the SERVICE_OBJECTIVE setting determines the service level and the performance tier for your data warehouse.

```
--Optimized for Elasticity
CREATE DATABASE myElasticSQLDW
WITH
(    SERVICE_OBJECTIVE = 'DW1000'
)
;

--Optimized for Compute
CREATE DATABASE myComputeSQLDW
WITH
(    SERVICE_OBJECTIVE = 'DW1000c'
)
;
```

Memory maximums

The performance tiers have different memory profiles, which translates into a different amount of memory per query. The Optimized for Compute performance tier provides 2.5x more memory per query than the Optimized for Elasticity performance tier. This extra memory helps the Optimized for Compute performance tier deliver its blazing fast performance. Additional memory per query also allows you to run more queries concurrently since queries can use lower resource classes.

Optimized for Elasticity

The service levels for the Optimized for Elasticity performance tier range from DW100 to DW6000.

SERVICE LEVEL	MAX CONCURRENT QUERIES	COMPUTE NODES	DISTRIBUTIONS PER COMPUTE NODE	MAX MEMORY PER DISTRIBUTION (MB)	MAX MEMORY PER DATA WAREHOUSE (GB)
DW100	4	1	60	400	24
DW200	8	2	30	800	48
DW300	12	3	20	1,200	72
DW400	16	4	15	1,600	96
DW500	20	5	12	2,000	120
DW600	24	6	10	2,400	144
DW1000	32	10	6	4,000	240
DW1200	32	12	5	4,800	288
DW1500	32	15	4	6,000	360
DW2000	32	20	3	8,000	480
DW3000	32	30	2	12,000	720
DW6000	32	60	1	24,000	1440

Optimized for Compute

The service levels for the Optimized for Compute performance tier range from DW1000c to DW30000c

SERVICE LEVEL	MAX CONCURRENT QUERIES	COMPUTE NODES	DISTRIBUTIONS PER COMPUTE NODE	MAX MEMORY PER DISTRIBUTION (GB)	MAX MEMORY PER DATA WAREHOUSE (GB)
DW1000c	32	2	30	10	600
DW1500c	32	3	20	15	900
DW2000c	32	4	15	20	1200
DW2500c	32	5	12	25	1500
DW3000c	32	6	10	30	1800
DW5000c	32	10	6	50	3000
DW6000c	32	12	5	60	3600
DW7500c	32	15	4	75	4500
DW10000c	32	20	3	100	6000
DW15000c	32	30	2	150	9000
DW30000c	32	60	1	300	18000

The maximum cDWU is DW30000c, which has 60 Compute nodes and one distribution per Compute node. For example, a 600 TB data warehouse at DW30000c processes approximately 10 TB per Compute node.

Concurrency maximums

SQL Data Warehouse provides industry-leading concurrency on a single data warehouse. To ensure each query has enough resources to execute efficiently, the system tracks compute resource utilization by assigning concurrency slots to each query. The system puts queries into a queue where they wait until enough concurrency slots are available.

Concurrency slots also determine CPU prioritization. For more information, see Analyze your workload

Concurrency slots

Concurrency slots are a convenient way to track the resources available for query execution. They are like tickets that you purchase to reserve seats at a concert because seating is limited. Similarly, SQL Data Warehouse has a finite number of compute resources. Queries reserve compute resources by acquiring concurrency slots. Before a query can start executing, it must be able to reserve enough concurrency slots. When a query finishes, it releases its concurrency slots.

- The optimized for elasticity performance tier scales to 240 concurrency slots.
- The optimized for compute performance tier scales to 1200 concurrency slots.

Each query will consume zero, one, or more concurrency slots. System queries and some trivial queries do not consume any slots. By default, queries that are governed by resource classes require one concurrency slot. More complex queries can require additional concurrency slots.

- A query running with 10 concurrency slots can access 5 times more compute resources than a query running with 2 concurrency slots.

Only resource governed queries consume concurrency slots. The exact number of concurrency slots consumed is determined by the query's resource class.

Optimized for Compute

The following table shows the maximum concurrent queries and concurrency slots for each dynamic resource class. These apply to the Optimized for Compute performance tier.

Dynamic resource classes

SERVICE LEVEL	MAXIMUM CONCURRENT QUERIES	CONCURRENCY SLOTS AVAILABLE	SLOTS USED BY SMALLRC	SLOTS USED BY MEDIUMRC	SLOTS USED BY LARGERC	SLOTS USED BY XLARGERC
DW1000c	32	40	1	8	16	32
DW1500c	32	60	1	8	16	32
DW2000c	32	80	1	16	32	64
DW2500c	32	100	1	16	32	64
DW3000c	32	120	1	16	32	64
DW5000c	32	200	1	32	64	128
DW6000c	32	240	1	32	64	128
DW7500c	32	300	1	64	128	128
DW10000c	32	400	1	64	128	256
DW15000c	32	600	1	64	128	256
DW30000c	32	1200	1	64	128	256

Static resource classes

The following table shows the maximum concurrent queries and concurrency slots for each static resource class.

SERVICE LEVEL	MAXIMUM CONCURRENT QUERIES	CONCURRENCY SLOTS AVAILABLE	STATIC RC10	STATIC RC20	STATIC RC30	STATIC RC40	STATIC RC50	STATIC RC60	STATIC RC70	STATIC RC80
DW1000c	32	40	1	2	4	8	16	32	32	32
DW1500c	32	60	1	2	4	8	16	32	64	64
DW2000c	32	80	1	2	4	8	16	32	64	64

SERVICE LEVEL	MAXIMUM CONCURRENT QUERIES	CONCURRENCY SLOTS AVAILABLE	STATIC RC10	STATIC RC20	STATIC RC30	STATIC RC40	STATIC RC50	STATIC RC60	STATIC RC70	STATIC RC80
DW2500c	32	100	1	2	4	8	16	32	64	64
DW3000c	32	120	1	2	4	8	16	32	64	128
DW5000c	32	200	1	2	4	8	16	32	64	128
DW6000c	32	240	1	2	4	8	16	32	64	128
DW7500c	32	300	1	2	4	8	16	32	64	128
DW10000c	32	400	1	2	4	8	16	32	64	128
DW15000c	32	600	1	2	4	8	16	32	64	128
DW30000c	32	1200	1	2	4	8	16	32	64	128

Optimized for Elasticity

The following table shows the maximum concurrent queries and concurrency slots for each dynamic resource class. These apply to the Optimized for Elasticity performance tier.

Dynamic resource classes

SERVICE LEVEL	MAXIMUM CONCURRENT QUERIES	CONCURRENCY SLOTS AVAILABLE	SMALLRC	MEDIUMRC	LARGERC	XLARGERC
DW100	4	4	1	1	2	4
DW200	8	8	1	2	4	8
DW300	12	12	1	2	4	8
DW400	16	16	1	4	8	16
DW500	20	20	1	4	8	16
DW600	24	24	1	4	8	16
DW1000	32	40	1	8	16	32
DW1200	32	48	1	8	16	32

SERVICE LEVEL	MAXIMUM CONCURRENT QUERIES	CONCURRENCY SLOTS AVAILABLE	SMALLRC	MEDIUMRC	LARGERC	XLARGERC
DW1500	32	60	1	8	16	32
DW2000	32	80	1	16	32	64
DW3000	32	120	1	16	32	64
DW6000	32	240	1	32	64	128

Static resource classes The following table shows the maximum concurrent queries and concurrency slots for each static resource class. These apply to the Optimized for Elasticity performance tier.

SERVICE LEVEL	MAXIMUM CONCURRENT QUERIES	MAXIMUM CONCURRENCY SLOTS	STATICRC10	STATICRC20	STATICRC30	STATICRC40	STATICRC50	STATICRC60	STATICRC70	STATICRC80
DW100	4	4	1	2	4	4	4	4	4	4
DW200	8	8	1	2	4	8	8	8	8	8
DW300	12	12	1	2	4	8	8	8	8	8
DW400	16	16	1	2	4	8	16	16	16	16
DW500	20	20	1	2	4	8	16	16	16	16
DW600	24	24	1	2	4	8	16	16	16	16
DW1000	32	40	1	2	4	8	16	32	32	32
DW1200	32	48	1	2	4	8	16	32	32	32
DW1500	32	60	1	2	4	8	16	32	32	32
DW2000	32	80	1	2	4	8	16	32	64	64
DW3000	32	120	1	2	4	8	16	32	64	64
DW6000	32	240	1	2	4	8	16	32	64	128

queries finishes and the number of queries and slots fall below the limits, SQL Data Warehouse releases queued queries.

Next steps

To learn more about how to leverage resource classes to optimize your workload further please review the following articles:

- Resource classes for workload management
- Analyzing your workload

Data Warehouse Units (DWUs) and compute Data Warehouse Units (cDWUs)

11/13/2017 • 6 min to read • Edit Online

Explains Data Warehouse Units (DWUs) and compute Data Warehouse Units (cDWUS) for Azure SQL Data Warehouse. Include recommendations on choosing the ideal number of data warehouse units, and how to change the number of them.

What are Data Warehouse Units?

With SQL Data Warehouse CPU, memory, and IO are bundled into units of compute scale called Data Warehouse Units (DWUs). A DWU represents an abstract, normalized measure of compute resources and performance. By changing your service level you alter the number of DWUs that are allocated to the system, which in turn adjusts the performance, and the cost, of your system.

To pay for higher performance, you can increase the number of data warehouse units. To pay 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 can a standard data warehousing query scan a large number of rows and then perform a complex aggregation? This operation is I/O and CPU intensive.
- How fast can the data warehouse ingest data from Azure Storage Blobs or Azure Data Lake? This operation is network and CPU intensive.
- How fast can the CREATE TABLE AS SELECT T-SQL command 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.

Performance Tiers and Data Warehouse Units

Each performance tier uses a slightly different unit of measure for their data warehouse units. This difference is reflected on the invoice as the unit of scale directly translates to billing.

- The optimized for elasticity performance tier is measured in Data Warehouse Units (DWUs).
- The optimized for compute performance tier is measured in compute Data Warehouse Units (cDWUs).

Both DWUs and cDWUs support scaling compute up or down, and pausing compute when you don't need to use the data warehouse. These operations are all on-demand. The optimized for compute performance tier also uses a local disk-based cache on the compute nodes to improve performance. When you scale or pause the system, the cache is invalidated and so a period of cache warming is required before optimal performance is achieved.

As you increase data warehouse units, you are linearly increasing computing resources. The optimized for compute performance tier provides the best query performance and highest scale but has a higher entry price. It is designed

Each SQL server (for example, myserver.database.windows.net) has a Database Transaction Unit (DTU) quota that allows a specific number of data warehouse units. For more information, see the workload management capacity limits.

How many data warehouse units do I 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. During development, begin by selecting a smaller DWU using the optimized for elasticity performance tier. Since the concern at this stage is functional validation, the Optimized for Elasticity performance tier is a reasonable option. A good starting point is DW200.
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. If the workload shows significant peaks and troughs in activity and there is a good reason to scale frequently, then favor the optimized for elasticity performance tier.
4. If you need more than 1000 DWU, then favor the Optimized for Compute performance tier since this gives the best performance.

SQL Data Warehouse 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.

> **NOTE**
> Query performance only increases with more parallelization if the work can be split between compute nodes. If you find that scaling is not changing your performance, you may need to tune your table design and/or your queries. For query tuning guidance,refer to the following performance articles.

Permissions

Changing the data warehouse units requires the permissions described in ALTER DATABASE.

View current DWU settings

To view the current DWU setting:

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

Azure portal

To change DWUs or cDWUs:

1. Open the Azure portal, open your database, and click **Scale**.
2. Under **Scale**, move the slider left or right to change the DWU setting.
3. Click **Save**. A confirmation message appears. Click **yes** to confirm or **no** to cancel.

PowerShell

To change the DWUs or cDWUs, use the [Set-AzureRmSqlDatabase][Set-AzureRmSqlDatabase] PowerShell cmdlet. The following example sets the service level objective to DW1000 for the database MySQLDW that is hosted on server MyServer.

```
Set-AzureRmSqlDatabase -DatabaseName "MySQLDW" -ServerName "MyServer" -RequestedServiceObjectiveName "DW1000"
```

T-SQL

With T-SQL you can view the current DWU or cDWU settings, change the settings, and check the progress.

To change the DWUs or cDWUs:

1. Connect to the master database associated with your logical SQL Database server.
2. Use the ALTER DATABASE TSQL statement. The following example sets the service level objective to DW1000 for the database MySQLDW.

```
ALTER DATABASE MySQLDW
MODIFY (SERVICE_OBJECTIVE = 'DW1000')
;
```

REST APIs

To change the DWUs, use the [Create or Update Database][Create or Update Database] REST API. The following example sets the service level objective to DW1000 for the database MySQLDW which is hosted on server MyServer. The server is in an Azure resource group named ResourceGroup1.

```
PUT https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group-
name}/providers/Microsoft.Sql/servers/{server-name}/databases/{database-name}?api-version=2014-04-01-preview
HTTP/1.1
Content-Type: application/json; charset=UTF-8

{
    "properties": {
        "requestedServiceObjectiveName": DW1000
    }
}
```

Check status of DWU changes

DWU changes may take several minutes to complete. If you are scaling automatically, consider implementing logic to ensure that certain operations have been completed before proceeding with another action.

Checking the database state through various endpoints allows you to correctly implement automation. The portal provides notification upon completion of an operation and the databases current state but does not allow for programmatic checking of state.

To check the status of DWU changes:

1. Connect to the master database associated with your logical SQL Database server.
2. Submit the following query to check database state.

```
SELECT    *
FROM      sys.databases
;
```

1. Submit the following query to check status of operation

```
SELECT    *
FROM      sys.dm_operation_status
WHERE     resource_type_desc = 'Database'
AND       major_resource_id = 'MySQLDW'
;
```

This DMV returns information about various management operations on your SQL Data Warehouse such as the operation and the state of the operation, which is either be IN_PROGRESS or COMPLETED.

The scaling workflow

When you initiate a scale operation, the system first kills all open sessions, rolling back any open transactions to ensure a consistent state. For scale operations, scaling only occurs after this transactional rollback has completed.

- For a scale-up operation, the system provisions the additional compute and then reattaches to the storage layer.
- For a scale-down operation, the unneeded nodes detach from the storage and reattach to the remaining nodes.

Next steps

Refer to the following articles to help you understand some additional key performance concepts:

- Workload and concurrency management
- Table design overview
- Table distribution
- Table indexing
- Table partitioning
- Table statistics
- Best practices

Backup and restore in SQL Data Warehouse

10/24/2017 • 3 min to read • Edit Online

This article explains the specifics of backups in SQL Data Warehouse. Use data warehouse backups to restore a database snapshot to the primary region, or restore a geo-backup to your geo-paired region.

What is a data warehouse backup?

A data warehouse backup is the copy of your database that you can use to restore a data warehouse. Since SQL Data Warehouse is a distributed system, a data warehouse backup consists of many files that are located in Azure storage. A data warehouse backup includes both local database snapshots and geo-backups of all the databases and files that are associated with a data warehouse.

Local snapshot backups

SQL Data Warehouse takes snapshots of your data warehouse throughout the day. Snapshots are available for seven days. SQL Data Warehouse supports an eight hour recovery point objective (RPO). You can restore your data warehouse in the primary region to any one of the snapshots taken in the past seven days.

To see when the last snapshot started, run this query on your online SQL Data Warehouse.

```
select   top 1 *
from     sys.pdw_loader_backup_runs
order by run_id desc
;
```

Geo-backups

SQL Data Warehouse performs a geo-backup once per day to a paired data center. The RPO for a geo-restore is 24 hours. You can restore the geo-backup to the server in the geo-paired region. geo-backup ensures you can restore data warehouse in case you cannot access the snapshots in your primary region.

Geo-backups are on by default. If your data warehouse is optimized for elasticity, you can opt out if you wish. You cannot opt out of geo-backups with the optimized for compute performance tier.

Backup costs

You will notice the Azure bill has a line item for Azure Premium Storage and a line item for geo-redundant storage. The Premium Storage charge is the total cost for storing your data in the primary region, which includes snapshots. The geo-redundant charge covers the cost for storing the geo-backups.

The total cost for your primary data warehouse and seven days of Azure Blob snapshots is rounded to the nearest TB. For example, if your data warehouse is 1.5 TB and the snapshots use 100 GB, you are billed for 2 TB of data at Azure Premium Storage rates.

> **NOTE**
>
> Each snapshot is empty initially and grows as you make changes to the primary data warehouse. All snapshots increase in size as the data warehouse changes. Therefore, the storage costs for snapshots grow according to the rate of change.

at the standard Read-Access Geographically Redundant Storage (RA-GRS) rate.

For more information about SQL Data Warehouse pricing, see SQL Data Warehouse Pricing.

Snapshot retention when a data warehouse is paused

SQL Data Warehouse does not create snapshots and does not expire snapshots while a data warehouse is paused. The snapshot age does not change while the data warehouse is paused. Snapshot retention is based on the number of days the data warehouse is online, not calendar days.

For example, if a snapshot starts October 1 at 4 pm and the data warehouse is paused October 3 at 4 pm, the snapshots are up to two days old. When the data warehouse comes back online the snapshot is two days old. If the data warehouse comes online October 5 at 4 pm, the snapshot is two days old and remains for five more days.

When the data warehouse comes back online, SQL Data Warehouse resumes new snapshots and expires snapshots when they have more than seven days of data.

Can I restore a dropped data warehouse?

When you drop a data warehouse, SQL Data Warehouse creates a final snapshot and saves it for seven days. You can restore the data warehouse to the final restore point created at deletion.

> **IMPORTANT**
> If you delete a logical SQL server instance, all databases that belong to the instance are also deleted and cannot be recovered. You cannot restore a deleted server.

Next steps

To restore a SQL data warehouse, see Restore a SQL data warehouse.

For a business continuity overview, see Business continuity overview

Auditing in Azure SQL Data Warehouse

1/17/2018 • 4 min to read • Edit Online

SQL Data Warehouse auditing allows you to record events in your database to an audit log in your Azure Storage account. Auditing can help you maintain regulatory compliance, understand database activity, and gain insight into discrepancies and anomalies that could indicate business concerns or suspected security violations. SQL Data Warehouse auditing also integrates with Microsoft Power BI for reporting and analysis.

Auditing tools enable and facilitate adherence to compliance standards but don't guarantee compliance. For more information about Azure programs that support standards compliance, see the Azure Trust Center.

Auditing basics

SQL Data Warehouse database auditing allows you to:

- **Retain** an audit trail of selected events. You can define categories of database actions to be audited.
- **Report** on database activity. You can use preconfigured reports and a dashboard to get started quickly with activity and event reporting.
- **Analyze** reports. You can find suspicious events, unusual activity, and trends.

You can configure auditing for the following event categories:

Plain SQL and **Parameterized SQL** for which the collected audit logs are classified as

- **Access to data**
- **Schema changes (DDL)**
- **Data changes (DML)**
- **Accounts, roles, and permissions (DCL)**
- **Stored Procedure**, **Login** and, **Transaction Management**.

For each Event Category, Auditing of **Success** and **Failure** operations are configured separately.

For more information about the activities and events audited, see the Audit Log Format Reference (doc file download).

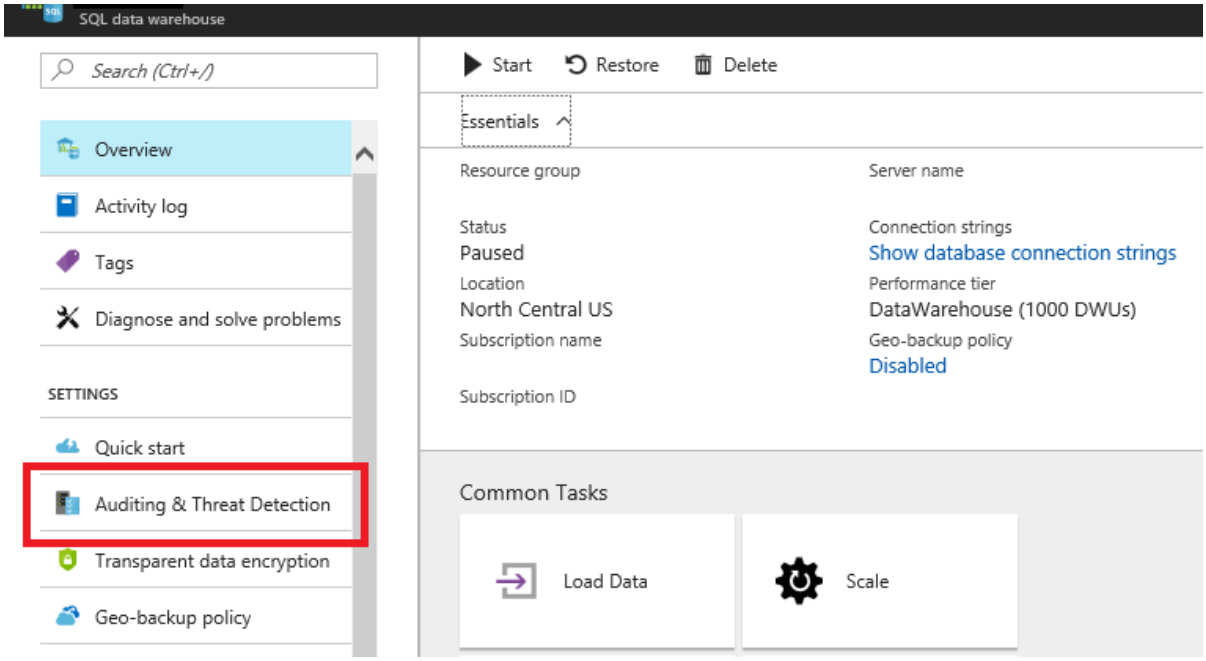
Audit logs are stored in your Azure storage account. You can define an audit log retention period.

You can define an auditing policy for a specific database or as a default server policy. A default server auditing policy applies to all databases on a server that do not have a specific overriding database auditing policy defined.

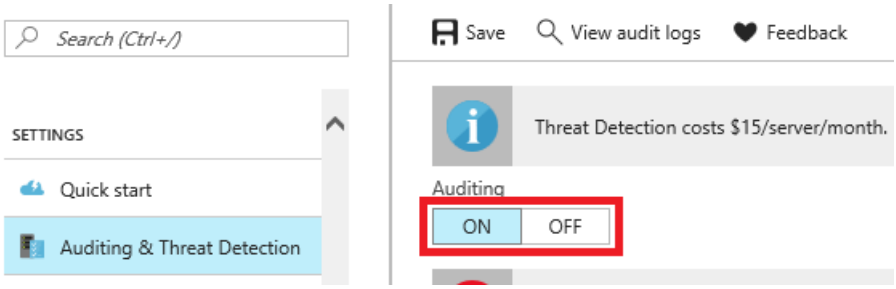
Before setting up audit auditing check if you are using a "Downlevel Client."

Set up auditing for your database

1. Launch the Azure portal.
2. Go to **Settings** for the SQL Data Warehouse you want to audit. Select **Auditing & Threat detection**.

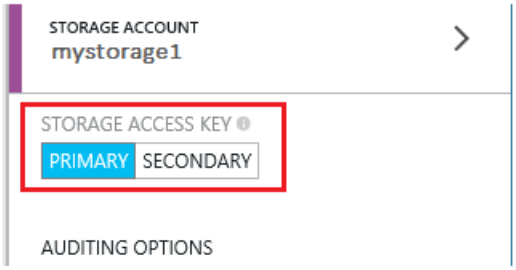


3. Next, enable auditing by clicking the **ON** button.



4. In the auditing configuration panel, select **STORAGE DETAILS** to open the Audit Logs Storage panel. Select the Azure storage account for the logs, and the retention period.

> **TIP**
> Use the same storage account for all audited databases to get the most out of the pre-configured reports templates.



5. Click the **OK** button to save the storage details configuration.
6. Under **LOGGING BY EVENT**, click **SUCCESS** and **FAILURE** to log all events, or choose individual event categories.
7. If you are configuring Auditing for a database, you may need to alter the connection string of your client to ensure data auditing is correctly captured. Check the Modify Server FDQN in the connection string topic for downlevel client connections.
8. Click **OK**.

Analyze audit logs and reports

Audit logs are aggregated in a collection of Store Tables with a **SQLDBAuditLogs** prefix in the Azure storage account you chose during setup. You can view log files using a tool such as Azure Storage Explorer.

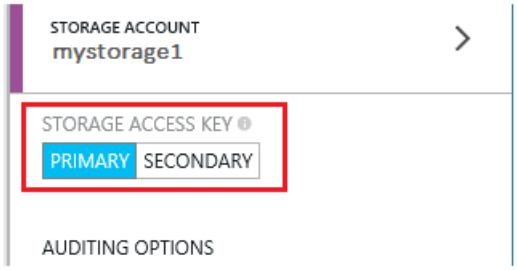
A preconfigured dashboard report template is available as a downloadable Excel spreadsheet to help you quickly analyze log data. To use the template on your audit logs, you need Excel 2013 or later and Power Query, which you can download here.

The template has fictional sample data in it, and you can set up Power Query to import your audit log directly from your Azure storage account.

Storage key regeneration

In production, you are likely to refresh your storage keys periodically. When refreshing your keys, you need to save the policy. The process is as follows:

1. In auditing configuration panel, which is described in the preceding setup auditing section, change the **Storage Access Key** from *Primary* to *Secondary* and **SAVE**.



2. Go to the storage configuration panel and **regenerate** the *Primary Access Key*.
3. Go back to the auditing configuration panel,
4. switch the **Storage Access Key** from *Secondary* to *Primary* and press **SAVE**.
5. Go back to the storage UI and **regenerate** the *Secondary Access Key* (as preparation for the next keys refresh cycle.

Automation (PowerShell/REST API)

You can also configure auditing in Azure SQL Data Warehouse by using the following automation tools:

- **PowerShell cmdlets**:
  - Get-AzureRMSqlDatabaseAuditingPolicy
  - Get-AzureRMSqlServerAuditingPolicy
  - Remove-AzureRMSqlDatabaseAuditing
  - Remove-AzureRMSqlServerAuditing
  - Set-AzureRMSqlDatabaseAuditingPolicy
  - Set-AzureRMSqlServerAuditingPolicy
  - Use-AzureRMSqlServerAuditingPolicy

Downlevel clients support for auditing and dynamic data masking

Auditing works with SQL clients that support TDS redirection.

Any client that implements TDS 7.4 should also support redirection. Exceptions to this include JDBC 4.0 in which the redirection feature is not fully supported and Tedious for Node.JS in which redirection was not implemented.

as follows:

- Original server FQDN in the connection string: <*server name*>.database.windows.net
- Modified server FQDN in the connection string: <*server name*>.database.**secure**.windows.net

A partial list of "Downlevel clients" includes:

- .NET 4.0 and below,
- ODBC 10.0 and below.
- JDBC (while JDBC does support TDS 7.4, the TDS redirection feature is not fully supported)
- Tedious (for Node.JS)

Remark: The preceding server FDQN modification may be useful also for applying a SQL Server Level Auditing policy without a need for a configuration step in each database (Temporary mitigation).

SQL Data Warehouse capacity limits

12/15/2017 • 5 min to read • Edit Online

The following tables contain the maximum values allowed for various components of Azure SQL Data Warehouse.

Workload management

CATEGORY	DESCRIPTION	MAXIMUM
Data Warehouse Units (DWU)	Max DWU for a single SQL Data Warehouse	Optimized for Elasticity performance tier: DW6000 Optimized for Compute performance tier: DW30000c
Data Warehouse Units (DWU)	Default DTU per server	54,000 By default, each SQL server (for example, myserver.database.windows.net) has a DTU Quota of 54,000, which allows up to DW6000c. This quota is simply a safety limit. You can increase your quota by creating a support ticket and selecting *Quota* as the request type. To calculate your DTU needs, multiply the 7.5 by the total DWU needed, or multiply 9.0 by the total cDWU needed. For example: DW6000 x 7.5 = 45,000 DTUs DW600c x 9.0 = 54,000 DTUs. You can view your current DTU consumption from the SQL server option in the portal. Both paused and unpaused databases count toward the DTU quota.
Database connection	Concurrent open sessions	1024 Each of the 1024 active sessions can submit requests to a SQL Data Warehouse database at the same time. Note, there are limits on the number of queries that can execute concurrently. When the concurrency limit is exceeded, the request goes into an internal queue where it waits to be processed.
Database connection	Maximum memory for prepared statements	20 MB

CATEGORY	DESCRIPTION	MAXIMUM
Workload management	Maximum concurrent queries	32 By default, SQL Data Warehouse can execute a maximum of 32 concurrent queries and queues remaining queries. The number of concurrent queries can descrease when users are assigned to higher resource classes or when SQL Data Warehouse has a lower service level. Some queries, like DMV queries, are always allowed to run.
tempdb	Maximum GB	399 GB per DW100. Therefore at DWU1000, tempdb is sized to 3.99 TB

Database objects

CATEGORY	DESCRIPTION	MAXIMUM
Database	Max size	240 TB compressed on disk This space is independent of tempdb or log space, and therefore this space is dedicated to permanent tables. Clustered columnstore compression is estimated at 5X. This compression allows the database to grow to approximately 1 PB when all tables are clustered columnstore (the default table type).
Table	Max size	60 TB compressed on disk
Table	Tables per database	2 billion
Table	Columns per table	1024 columns
Table	Bytes per column	Dependent on column data type. Limit is 8000 for char data types, 4000 for nvarchar, or 2 GB for MAX data types.
Table	Bytes per row, defined size	8060 bytes The number of bytes per row is calculated in the same manner as it is for SQL Server with page compression. Like SQL Server, SQL Data Warehouse supports row-overflow storage, which enables **variable length columns** to be pushed off-row. When variable length rows are pushed off-row, only 24-byte root is stored in the main record. For more information, see the Row-Overflow Data Exceeding 8-KB.

CATEGORY	DESCRIPTION	MAXIMUM
Table	Partitions per table	15,000 For high performance, we recommend minimizing the number of partitions you need while still supporting your business requirements. As the number of partitions grows, the overhead for Data Definition Language (DDL) and Data Manipulation Language (DML) operations grows and causes slower performance.
Table	Characters per partition boundary value.	4000
Index	Non-clustered indexes per table.	999 Applies to rowstore tables only.
Index	Clustered indexes per table.	1 Applies to both rowstore and columnstore tables.
Index	Index key size.	900 bytes. Applies to rowstore indexes only. Indexes on varchar columns with a maximum size of more than 900 bytes can be created if the existing data in the columns does not exceed 900 bytes when the index is created. However, later INSERT or UPDATE actions on the columns that cause the total size to exceed 900 bytes will fail.
Index	Key columns per index.	16 Applies to rowstore indexes only. Clustered columnstore indexes include all columns.
Statistics	Size of the combined column values.	900 bytes.
Statistics	Columns per statistics object.	32
Statistics	Statistics created on columns per table.	30,000
Stored Procedures	Maximum levels of nesting.	8
View	Columns per view	1,024

Loads

CATEGORY	DESCRIPTION	MAXIMUM
Polybase Loads	MB per row	1 Polybase loads only to rows that are smaller than 1 MB, and cannot load to VARCHAR(MAX), NVARCHAR(MAX) or VARBINARY(MAX).

Queries

CATEGORY	DESCRIPTION	MAXIMUM
Query	Queued queries on user tables.	1000
Query	Concurrent queries on system views.	100
Query	Queued queries on system views	1000
Query	Maximum parameters	2098
Batch	Maximum size	65,536*4096
SELECT results	Columns per row	4096 You can never have more than 4096 columns per row in the SELECT result. There is no guarantee that you can always have 4096. If the query plan requires a temporary table, the 1024 columns per table maximum might apply.
SELECT	Nested subqueries	32 You can never have more than 32 nested subqueries in a SELECT statement. There is no guarantee that you can always have 32. For example, a JOIN can introduce a subquery into the query plan. The number of subqueries can also be limited by available memory.
SELECT	Columns per JOIN	1024 columns You can never have more than 1024 columns in the JOIN. There is no guarantee that you can always have 1024. If the JOIN plan requires a temporary table with more columns than the JOIN result, the 1024 limit applies to the temporary table.

CATEGORY	DESCRIPTION	MAXIMUM
SELECT	Bytes per GROUP BY columns.	8060 The columns in the GROUP BY clause can have a maximum of 8060 bytes.
SELECT	Bytes per ORDER BY columns	8060 bytes. The columns in the ORDER BY clause can have a maximum of 8060 bytes.
Identifiers and constants per statement	Number of referenced identifiers and constants.	65,535 SQL Data Warehouse limits the number of identifiers and constants that can be contained in a single expression of a query. This limit is 65,535. Exceeding this number results in SQL Server error 8632. For more information, see Internal error: An expression services limit has been reached.

Metadata

SYSTEM VIEW	MAXIMUM ROWS
sys.dm_pdw_component_health_alerts	10,000
sys.dm_pdw_dms_cores	100
sys.dm_pdw_dms_workers	Total number of DMS workers for the most recent 1000 SQL requests.
sys.dm_pdw_errors	10,000
sys.dm_pdw_exec_requests	10,000
sys.dm_pdw_exec_sessions	10,000
sys.dm_pdw_request_steps	Total number of steps for the most recent 1000 SQL requests that are stored in sys.dm_pdw_exec_requests.
sys.dm_pdw_os_event_logs	10,000
sys.dm_pdw_sql_requests	The most recent 1000 SQL requests that are stored in sys.dm_pdw_exec_requests.

Next steps

For more reference information, see SQL Data Warehouse reference overview.

SQL Data Warehouse Frequently asked questions

6/27/2017 • 2 min to read • Edit Online

General

Q. What does SQL DW offer for data security?

A. SQL DW offers several solutions for protecting data such as TDE and auditing. For more information, see Security.

Q. Where can I find out what legal or business standards is SQL DW compliant with?

A. Visit the Microsoft Compliance page for various compliance offerings by product such as SOC and ISO. First choose by Compliance title, then expand Azure in the Microsoft in-scope cloud services section on the right side of the page to see what services are Azure services are compliant.

Q. Can I connect PowerBI?

A. Yes! Though PowerBI supports direct query with SQL DW, it's not intended for large number of users or real-time data. For production use of PowerBI, we recommend using PowerBI on top of Azure Analysis Services or Analysis Service IaaS.

Q. What are SQL Data Warehouse Capacity Limits?

A. See our current capacity limits page.

Q. Why is my Scale/Pause/Resume taking so long?

A. A variety of factors can influence the time for compute management operations. A common case for long running operations is transactional rollback. When a scale or pause operation is initiated, all incoming sessions are blocked and queries are drained. In order to leave the system in a stable state, transactions must be rolled back before an operation can commence. The greater the number and larger the log size of transactions, the longer the operation will be stalled restoring the system to a stable state.

User support

Q. I have a feature request, where do I submit it?

A. If you have a feature request, submit it on our UserVoice page

Q. How can I do x?

A. For help in developing with SQL Data Warehouse, you can ask questions on our Stack Overflow page.

Q. How do I submit a support ticket?

A. Support Tickets can be filed through Azure portal.

SQL language/feature support

Q. What datatypes does SQL Data Warehouse support?

A. See SQL Data Warehouse data types.

Q. What table features do you support?

Unsupported Table Features.

Tooling and administration

Q. Do you support Database projects in Visual Studio.

A. We currently do not support Database projects in Visual Studio for SQL Data Warehouse. If you'd like to cast a vote to get this feature, visit our User Voice Database projects feature request.

Q. Does SQL Data Warehouse support REST APIs?

A. Yes. Most REST functionality that can be used with SQL Database is also available with SQL Data Warehouse. You can find API information within REST documentation pages or MSDN.

Loading

Q. What client drivers do you support?

A. Driver support for DW can be found on the Connection Strings page

Q: What file formats are supported by PolyBase with SQL Data Warehouse?

A: Orc, RC, Parquet, and flat delimited text

Q: What can I connect to from SQL DW using PolyBase?

A: Azure Data Lake Store and Azure Storage Blobs

Q: Is computation pushdown possible when connecting to Azure Storage Blobs or ADLS?

A: No, SQL DW PolyBase only interacts the storage components.

Q: Can I connect to HDI?

A: HDI can use either ADLS or WASB as the HDFS layer. If you have either as your HDFS layer, then you can load that data into SQL DW. However, you cannot generate pushdown computation to the HDI instance.

Next steps

For more information on SQL Data Warehouse as a whole, see our Overview page.

Secure a database in SQL Data Warehouse

1/2/2018 • 5 min to read • Edit Online

This article walks through the basics of securing your Azure SQL Data Warehouse database. In particular, this article will get you started with resources for limiting access, protecting data, and monitoring activities on a database.

Connection security

Connection Security refers to how you restrict and secure connections to your database using firewall rules and connection encryption.

Firewall rules are used by both the server and the database to reject connection attempts from IP addresses that have not been explicitly whitelisted. To allow connections from your application or client machine's public IP address, you must first create a server-level firewall rule using the Azure Portal, REST API, or PowerShell. As a best practice, you should restrict the IP address ranges allowed through your server firewall as much as possible. To access Azure SQL Data Warehouse from your local computer, ensure the firewall on your network and local computer allows outgoing communication on TCP port 1433. For more information, see Azure SQL Database firewall, sp_set_firewall_rule.

Connections to your SQL Data Warehouse are encrypted by default. Modifying connection settings to disable encryption are ignored.

Authentication

Authentication refers to how you prove your identity when connecting to the database. SQL Data Warehouse currently supports SQL Server Authentication with a username and password as well as a Azure Active Directory.

When you created the logical server for your database, you specified a "server admin" login with a username and password. Using these credentials, you can authenticate to any database on that server as the database owner, or "dbo" through SQL Server Authentication.

However, as a best practice, your organization's users should use a different account to authenticate. This way you can limit the permissions granted to the application and reduce the risks of malicious activity in case your application code is vulnerable to a SQL injection attack.

To create a SQL Server Authenticated user, connect to the **master** database on your server with your server admin login and create a new server login. Additionally, it is a good idea to create a user in the master database for Azure SQL Data Warehouse users. Creating a user in master allows a user to login using tools like SSMS without specifying a database name. It also allows them to use the object explorer to view all databases on a SQL server.

```
-- Connect to master database and create a login
CREATE LOGIN ApplicationLogin WITH PASSWORD = 'Str0ng_password';
CREATE USER ApplicationUser FOR LOGIN ApplicationLogin;
```

Then, connect to your **SQL Data Warehouse database** with your server admin login and create a database user based on the server login you just created.

```
-- Connect to SQL DW database and create a database user
CREATE USER ApplicationUser FOR LOGIN ApplicationLogin;
```

If a user will be doing additional operations such as creating logins or creating new databases, they will also need to be assigned to the `Loginmanager` and `dbmanager` roles in the master database. For more information on these additonal roles and authenticating to a SQL Database, see Managing databases and logins in Azure SQL Database. For more details on Azure AD for SQL Data Warehouse, see Connecting to SQL Data Warehouse By Using Azure Active Directory Authentication.

Authorization

Authorization refers to what you can do within an Azure SQL Data Warehouse database, and this is controlled by your user account's role memberships and permissions. As a best practice, you should grant users the least privileges necessary. Azure SQL Data Warehouse makes this easy to manage with roles in T-SQL:

```
EXEC sp_addrolemember 'db_datareader', 'ApplicationUser'; -- allows ApplicationUser to read data
EXEC sp_addrolemember 'db_datawriter', 'ApplicationUser'; -- allows ApplicationUser to write data
```

The server admin account you are connecting with is a member of db_owner, which has authority to do anything within the database. Save this account for deploying schema upgrades and other management operations. Use the "ApplicationUser" account with more limited permissions to connect from your application to the database with the least privileges needed by your application.

There are ways to further limit what a user can do with Azure SQL Data Warehouse:

- Granular Permissions let you control which operations you can do on individual columns, tables, views, schemas, procedures, and other objects in the database. Use granular permissions to have the most control and grant the minimum permissions necessary. The granular permission system is somewhat complicated and will require some study to use effectively.
- Database roles other than db_datareader and db_datawriter can be used to create more powerful application user accounts or less powerful management accounts. The built-in fixed database roles provide an easy way to grant permissions, but can result in granting more permissions than are necessary.
- Stored procedures can be used to limit the actions that can be taken on the database.

Below is an example of granting read access to a user-defined schema.

```
--CREATE SCHEMA Test
GRANT SELECT ON SCHEMA::Test to ApplicationUser
```

Managing databases and logical servers from the Azure portal or using the Azure Resource Manager API is controlled by your portal user account's role assignments. For more information on this topic, see Role-based access control in Azure Portal.

Encryption

Azure SQL Data Warehouse Transparent Data Encryption (TDE) helps protect against the threat of malicious activity by performing real-time encryption and decryption of your data at rest. When you encrypt your database, associated backups and transaction log files are encrypted without requiring any changes to your applications. TDE encrypts the storage of an entire database by using a symmetric key called the database encryption key. In SQL Database the database encryption key is protected by a built-in server certificate. The built-in server certificate is unique for each SQL Database server. Microsoft automatically rotates these certificates at least every 90 days. The encryption algorithm used by SQL Data Warehouse is AES-256. For a general description of TDE, see

You can encrypt your database using the Azure Portal or T-SQL.

Next steps

For details and examples on connecting to your SQL Data Warehouse with different protocols, see Connect to SQL Data Warehouse.

Authentication to Azure SQL Data Warehouse

7/3/2017 • 2 min to read • Edit Online

To connect to SQL Data Warehouse, you must pass in security credentials for authentication purposes. Upon establishing a connection, certain connection settings are configured as part of establishing your query session.

For more information on security and how to enable connections to your data warehouse, see Secure a database in SQL Data Warehouse.

SQL authentication

To connect to SQL Data Warehouse, you must provide the following information:

- Fully qualified servername
- Specify SQL authentication
- Username
- Password
- Default database (optional)

By default your connection connects to the *master* database and not your user database. To connect to your user database, you can choose to do one of two things:

- Specify the default database when registering your server with the SQL Server Object Explorer in SSDT, SSMS, or in your application connection string. For example, include the InitialCatalog parameter for an ODBC connection.
- Highlight the user database before creating a session in SSDT.

> **NOTE**
> The Transact-SQL statement **USE MyDatabase;** is not supported for changing the database for a connection. For guidance connecting to SQL Data Warehouse with SSDT, refer to the Query with Visual Studio article.

Azure Active Directory (AAD) authentication

Azure Active Directory authentication is a mechanism of connecting to Microsoft Azure SQL Data Warehouse by using identities in Azure Active Directory (Azure AD). With Azure Active Directory authentication, you can centrally manage the identities of database users and other Microsoft services in one central location. Central ID management provides a single place to manage SQL Data Warehouse users and simplifies permission management.

Benefits

Azure Active Directory benefits include:

- Provides an alternative to SQL Server authentication.
- Helps stop the proliferation of user identities across database servers.
- Allows password rotation in a single place
- Manage database permissions using external (AAD) groups.
- Eliminates storing passwords by enabling integrated Windows authentication and other forms of authentication supported by Azure Active Directory.

- Supports token-based authentication for applications connecting to SQL Data Warehouse.
- Supports Multi-Factor authentication through Active Directory Universal Authentication for SQL Server Management Studio. For a description of Multi-Factor Authentication, see SSMS support for Azure AD MFA with SQL Database and SQL Data Warehouse.

> **NOTE**
> Azure Active Directory is still relatively new and has some limitations. To ensure that Azure Active Directory is a good fit for your environment, see Azure AD features and limitations, specifically the Additional considerations.

Configuration steps

Follow these steps to configure Azure Active Directory authentication.

1. Create and populate an Azure Active Directory
2. Optional: Associate or change the active directory that is currently associated with your Azure Subscription
3. Create an Azure Active Directory administrator for Azure SQL Data Warehouse.
4. Configure your client computers
5. Create contained database users in your database mapped to Azure AD identities
6. Connect to your data warehouse by using Azure AD identities

Currently Azure Active Directory users are not shown in SSDT Object Explorer. As a workaround, view the users in sys.database_principals.

Find the details

- The steps to configure and use Azure Active Directory authentication are nearly identical for Azure SQL Database and Azure SQL Data Warehouse. Follow the detailed steps in the topic Connecting to SQL Database or SQL Data Warehouse By Using Azure Active Directory Authentication.
- Create custom database roles and add users to the roles. Then grant granular permissions to the roles. For more information, see Getting Started with Database Engine Permissions.

Next steps

To start querying your data warehouse with Visual Studio and other applications, see Query with Visual Studio.

Migrate your solution to Azure SQL Data Warehouse

6/30/2017 • 2 min to read • Edit Online

See what's involved in migrating an existing database solution to Azure SQL Data Warehouse.

Profile your workload

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 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, 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.

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.

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

- High frequency reads and writes
- Large numbers of singleton selects
- High volumes of single row inserts
- Row by row processing needs
- Incompatible formats (JSON, XML)

Plan the migration

Once you have decided to migrate an existing solution to SQL Data Warehouse, it is important to plan the migration before getting started.

One goal of planning is to ensure your data, your table schemas, and your code are compatible with SQL Data Warehouse. There are some compatibility differences to work around between your current system and SQL Data Warehouse. Plus, migrating large amounts of data to Azure takes time. Careful planning expedites getting your data to Azure.

Another goal of planning is to make design adjustments to ensure your solution takes advantage of the high query performance SQL Data Warehouse is designed to provide. Designing data warehouses for scale introduces different design patterns and so traditional approaches aren't always the best. Although you can make some design adjustments after migration, making changes sooner in the process will save time later.

To perform a successful migration, you need to migrate your table schemas, your code, and your data. For guidance on these migration topics, see:

- Migrate your schemas
- Migrate your code
- Migrate your data.

Next steps

through blogs. Take a look at their article, Migrating data to Azure SQL Data Warehouse in practice for additional guidance on migration.

Migrate your schemas to SQL Data Warehouse

6/30/2017 • 3 min to read • Edit Online

Guidance for migrating your SQL schemas to SQL Data Warehouse.

Plan your schema migration

As you plan a migration, see the table overview to become familiar with table design considerations such as statistics, distribution, partitioning, and indexing. It also lists some unsupported table features and their workarounds.

Use user-defined schemas to consolidate databases

Your existing workload probably has more than one database. For example, a SQL Server data warehouse might include a staging database, a data warehouse database, and some data mart databases. In this topology, each database runs as a separate workload with separate security policies.

By contrast, SQL Data Warehouse runs the entire data warehouse workload within one database. Cross database joins are not permitted. Therefore, SQL Data Warehouse expects all tables used by the data warehouse to be stored within the one database.

We recommend using user-defined schemas to consolidate your existing workload into one database. For examples, see User-defined schemas

Use compatible data types

Modify your data types to be compatible with SQL Data Warehouse. For a list of supported and unsupported data types, see data types. That topic gives workarounds for the unsupported types. It also provides a query to identify existing types that are not supported in SQL Data Warehouse.

Minimize row size

For best performance, minimize the row length of your tables. Since shorter row lengths lead to better performance, use the smallest data types that work for your data.

For table row width, PolyBase has a 1 MB limit. If you plan to load data into SQL Data Warehouse with PolyBase, update your tables to have maximum row widths of less than 1 MB.

Specify the distribution option

SQL Data Warehouse is a distributed database system. Each table is distributed or replicated across the Compute nodes. There's a table option that lets you specify how to distribute the data. The choices are round-robin, replicated, or hash distributed. Each has pros and cons. If you don't specify the distribution option, SQL Data Warehouse will use round-robin as the default.

- Round-robin is the default. It is the simplest to use, and loads the data as fast as possible, but joins will require data movement which slows query performance.
- Replicated stores a copy of the table on each Compute node. Replicated tables are performant because they do not require data movement for joins and aggregations. They do require extra storage, and therefore work best for smaller tables.
- Hash-distributed distributes the rows across all the nodes via a hash function. Hash-distributed tables are the

option requires some planning to select the best column on which to distribute the data. However, if you don't choose the best column the first time, you can easily re-distribute the data on a different column.

To choose the best distribution option for each table, see Distributed tables.

Next steps

Once you have successfully migrated your database schema to SQL Data Warehouse, proceed to one of the following articles:

- Migrate your data
- Migrate your code

For more about SQL Data Warehouse best practices, see the best practices article.

Migrate your SQL code to SQL Data Warehouse

6/30/2017 • 3 min to read • Edit Online

This article explains code changes you will probably need to make when migrating your code from another database to SQL Data Warehouse. Some SQL Data Warehouse features can significantly improve performance as they are designed to work in a distributed fashion. However, to maintain performance and scale, some features are also not available.

Common T-SQL Limitations

The following list summarizes the most common features that SQL Data Warehouse does not support. The links take you to workarounds for the unsupported features:

- ANSI joins on updates
- ANSI joins on deletes
- merge statement
- cross-database joins
- cursors
- INSERT..EXEC
- output clause
- inline user-defined functions
- multi-statement functions
- common table expressions
- [recursive common table expressions (CTE)](#Recursive-common-table-expressions-(CTE)
- CLR functions and procedures
- $partition function
- table variables
- table value parameters
- distributed transactions
- commit / rollback work
- save transaction
- execution contexts (EXECUTE AS)
- group by clause with rollup / cube / grouping sets options
- nesting levels beyond 8
- updating through views
- use of select for variable assignment
- no MAX data type for dynamic SQL strings

Fortunately most of these limitations can be worked around. Explanations are provided in the relevant development articles referenced above.

Supported CTE features

Common table expressions (CTEs) are partially supported in SQL Data Warehouse. The following CTE features are currently supported:

- A CTE can be specified in a SELECT statement.

- A CTE can be specified in a CREATE TABLE AS SELECT (CTAS) statement.
- A CTE can be specified in a CREATE REMOTE TABLE AS SELECT (CRTAS) statement.
- A CTE can be specified in a CREATE EXTERNAL TABLE AS SELECT (CETAS) statement.
- A remote table can be referenced from a CTE.
- An external table can be referenced from a CTE.
- Multiple CTE query definitions can be defined in a CTE.

CTE Limitations

Common table expressions have some limitations in SQL Data Warehouse including:

- A CTE must be followed by a single SELECT statement. INSERT, UPDATE, DELETE, and MERGE statements are not supported.
- A common table expression that includes references to itself (a recursive common table expression) is not supported (see below section).
- Specifying more than one WITH clause in a CTE is not allowed. For example, if a CTE_query_definition contains a subquery, that subquery cannot contain a nested WITH clause that defines another CTE.
- An ORDER BY clause cannot be used in the CTE_query_definition, except when a TOP clause is specified.
- When a CTE is used in a statement that is part of a batch, the statement before it must be followed by a semicolon.
- When used in statements prepared by sp_prepare, CTEs will behave the same way as other SELECT statements in PDW. However, if CTEs are used as part of CETAS prepared by sp_prepare, the behavior can defer from SQL Server and other PDW statements because of the way binding is implemented for sp_prepare. If SELECT that references CTE is using a wrong column that does not exist in CTE, the sp_prepare will pass without detecting the error, but the error will be thrown during sp_execute instead.

Recursive CTEs

Recursive CTEs are not supported in SQL Data Warehouse. The migration of recursive CTE can be somewhat complex and the best process is to break it into multiple steps. You can typically use a loop and populate a temporary table as you iterate over the recursive interim queries. Once the temporary table is populated you can then return the data as a single result set. A similar approach has been used to solve `GROUP BY WITH CUBE` in the group by clause with rollup / cube / grouping sets options article.

Unsupported system functions

There are also some system functions that are not supported. Some of the main ones you might typically find used in data warehousing are:

- NEWSEQUENTIALID()
- @@NESTLEVEL()
- @@IDENTITY()
- @@ROWCOUNT()
- ROWCOUNT_BIG
- ERROR_LINE()

Some of these issues can be worked around.

@@ROWCOUNT workaround

To work around lack of support for @@ROWCOUNT, create a stored procedure that will retrieve the last row count

```
CREATE PROCEDURE LastRowCount AS
WITH LastRequest as
(   SELECT TOP 1    request_id
    FROM            sys.dm_pdw_exec_requests
    WHERE           session_id = SESSION_ID()
    AND             resource_class IS NOT NULL
    ORDER BY end_time DESC
),
LastRequestRowCounts as
(
    SELECT  step_index, row_count
    FROM    sys.dm_pdw_request_steps
    WHERE   row_count >= 0
    AND     request_id IN (SELECT request_id from LastRequest)
)
SELECT TOP 1 row_count FROM LastRequestRowCounts ORDER BY step_index DESC
;
```

Next steps

For a complete list of all supported T-SQL statements, see Transact-SQL topics.

Migrate Your Data

9/25/2017 • 8 min to read • Edit Online

Data can be moved from different sources into your SQL Data Warehouse with a variety tools. ADF Copy, SSIS, and bcp can all be used to achieve this goal. However, as the amount of data increases you should think about breaking down the data migration process into steps. This affords you the opportunity to optimize each step both for performance and for resilience to ensure a smooth data migration.

This article first discusses the simple migration scenarios of ADF Copy, SSIS, and bcp. It then look a little deeper into how the migration can be optimized.

Azure Data Factory (ADF) copy

ADF Copy is part of Azure Data Factory. You can use ADF Copy to export your data to flat files residing on local storage, to remote flat files held in Azure blob storage or directly into SQL Data Warehouse.

If your data starts in flat files, then you will first need to transfer it to Azure storage blob before initiating a load it into SQL Data Warehouse. Once the data is transferred into Azure blob storage you can choose to use ADF Copy again to push the data into SQL Data Warehouse.

PolyBase also provides a high-performance option for loading the data. However, that does mean using two tools instead of one. If you need the best performance then use PolyBase. If you want a single tool experience (and the data is not massive) then ADF is your answer.

Head over to the following article for some great ADF samples.

Integration Services

Integration Services (SSIS) is a powerful and flexible Extract Transform and Load (ETL) tool that supports complex workflows, data transformation, and several data loading options. Use SSIS to simply transfer data to Azure or as part of a broader migration.

> **NOTE**
> SSIS can export to UTF-8 without the byte order mark in the file. To configure this you must first use the derived column component to convert the character data in the data flow to use the 65001 UTF-8 code page. Once the columns have been converted, write the data to the flat file destination adapter ensuring that 65001 has also been selected as the code page for the file.

SSIS connects to SQL Data Warehouse just as it would connect to a SQL Server deployment. However, your connections will need to be using an ADO.NET connection manager. You should also take care to configure the "Use bulk insert when available" setting to maximize throughput. Please refer to the ADO.NET destination adapter article to learn more about this property

> **NOTE**
> Connecting to Azure SQL Data Warehouse by using OLEDB is not supported.

In addition, there is always the possibility that a package might fail due to throttling or network issues. Design

bcp

bcp is a command-line utility that is designed for flat file data import and export. Some transformation can take place during data export. To perform simple transformations use a query to select and transform the data. Once exported, the flat files can then be loaded directly into the target the SQL Data Warehouse database.

> **NOTE**
> It is often a good idea to encapsulate the transformations used during data export in a view on the source system. This ensures that the logic is retained and the process is repeatable.

Advantages of bcp are:

- Simplicity. bcp commands are simple to build and execute
- Re-startable load process. Once exported the load can be executed any number of times

Limitations of bcp are:

- bcp works with tabulated flat files only. It does not work with files such as xml or JSON
- Data transformation capabilities are limited to the export stage only and are simple in nature
- bcp has not been adapted to be robust when loading data over the internet. Any network instability may cause a load error.
- bcp relies on the schema being present in the target database prior to the load

For more information, see Use bcp to load data into SQL Data Warehouse.

Optimizing data migration

A SQLDW data migration process can be effectively broken down into three discrete steps:

1. Export of source data
2. Transfer of data to Azure
3. Load into the target SQLDW database

Each step can be individually optimized to create a robust, re-startable and resilient migration process that maximizes performance at each step.

Optimizing data load

Looking at these in reverse order for a moment; the fastest way to load data is via PolyBase. Optimizing for a PolyBase load process places prerequisites on the preceding steps so it's best to understand this upfront. They are:

1. Encoding of data files
2. Format of data files
3. Location of data files

Encoding

PolyBase requires data files to be UTF-8 or UTF-16FE.

Format of data files

PolyBase mandates a fixed row terminator of \n or newline. Your data files must conform to this standard. There aren't any restrictions on string or column terminators.

exported columns are required and that the types conform to the required standards.

Please refer back to the [migrate your schema] article for detail on supported data types.

Location of data files

SQL Data Warehouse uses PolyBase to load data from Azure Blob Storage exclusively. Consequently, the data must have been first transferred into blob storage.

Optimizing data transfer

One of the slowest parts of data migration is the transfer of the data to Azure. Not only can network bandwidth be an issue but also network reliability can seriously hamper progress. By default migrating data to Azure is over the internet so the chances of transfer errors occurring are reasonably likely. However, these errors may require data to be re-sent either in whole or in part.

Fortunately you have several options to improve the speed and resilience of this process:

ExpressRoute

You may want to consider using ExpressRoute to speed up the transfer. ExpressRoute provides you with an established private connection to Azure so the connection does not go over the public internet. This is by no means a mandatory step. However, it will improve throughput when pushing data to Azure from an on-premises or co-location facility.

The benefits of using ExpressRoute are:

1. Increased reliability
2. Faster network speed
3. Lower network latency
4. higher network security

ExpressRoute is beneficial for a number of scenarios; not just the migration.

Interested? For more information and pricing please visit the ExpressRoute documentation.

Azure Import and Export Service

The Azure Import and Export Service is a data transfer process designed for large (GB++) to massive (TB++) transfers of data into Azure. It involves writing your data to disks and shipping them to an Azure data center. The disk contents will then be loaded into Azure Storage Blobs on your behalf.

A high-level view of the import export process is as follows:

1. Configure an Azure Blob Storage container to receive the data
2. Export your data to local storage
3. Copy the data to 3.5 inch SATA II/III hard disk drives using the [Azure Import/Export Tool]
4. Create an Import Job using the Azure Import and Export Service providing the journal files produced by the [Azure Import/Export Tool]
5. Ship the disks your nominated Azure data center
6. Your data is transferred to your Azure Blob Storage container
7. Load the data into SQLDW using PolyBase

AZCopy utility

The AZCopy utility is a great tool for getting your data transferred into Azure Storage Blobs. It is designed for small (MB++) to very large (GB++) data transfers. AZCopy has also been designed to provide good resilient throughput when transferring data to Azure and so is a great choice for the data transfer step. Once transferred you can load

using an "Execute Process" task.

To use AZCopy you will first need to download and install it. There is a production version and a preview version available.

To upload a file from your file system you will need a command like the one below:

```
AzCopy /Source:C:\myfolder /Dest:https://myaccount.blob.core.windows.net/mycontainer /DestKey:key
/Pattern:abc.txt
```

A high-level process summary could be:

1. Configure an Azure storage blob container to receive the data
2. Export your data to local storage
3. AZCopy your data in the Azure Blob Storage container
4. Load the data into SQL Data Warehouse using PolyBase

Full documentation available: AZCopy.

Optimizing data export

In addition to ensuring that the export conforms to the requirements laid out by PolyBase you can also seek to optimize the export of the data to improve the process further.

Data compression

PolyBase can read gzip compressed data. If you are able to compress your data to gzip files then you will minimize the amount of data being pushed over the network.

Multiple files

Breaking up large tables into several files not only helps to improve export speed, it also helps with transfer re-startability, and the overall manageability of the data once in the Azure blob storage. One of the many nice features of PolyBase is that it will read all the files inside a folder and treat it as one table. It is therefore a good idea to isolate the files for each table into its own folder.

PolyBase also supports a feature known as "recursive folder traversal". You can use this feature to further enhance the organization of your exported data to improve your data management.

To learn more about loading data with PolyBase, see Use PolyBase to load data into SQL Data Warehouse.

Next steps

For more about migration, see Migrate your solution to SQL Data Warehouse. For more development tips, see development overview.

Designing Extract, Load, and Transform (ELT) for Azure SQL Data Warehouse

12/13/2017 • 7 min to read • Edit Online

Combine the technologies for landing data in Azure storage and loading data into SQL Data Warehouse to design an Extract, Load, and Transform (ELT) process for Azure SQL Data Warehouse. This article introduces the technologies that support loading with Polybase, and then focuses on designing an ELT process that uses PolyBase with T-SQL to load data into SQL Data Warehouse from Azure Storage.

What is ELT?

Extract, Load, and Transform (ELT) is a process by which data moves from a source system to a destination data warehouse. This process is performed on a regular basis, for example hourly or daily, to get newly generated data into the data warehouse. The ideal way to get data from source to data warehouse is to develop an ELT process that uses PolyBase to load data into SQL Data Warehouse.

ELT loads first and then transforms the data, whereas Extract, Transform, and Load (ETL) transforms the data before loading it. Performing ELT instead of ETL saves the cost of providing your own resources to transform the data before it is loaded. When using SQL Data Warehouse, ELT leverages the MPP system to perform the transformations.

Although there are many variations for implementing ELT for SQL Data Warehouse, these are the basic steps:

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 SQL Data Warehouse staging tables by using PolyBase.
5. Transform the data.
6. Insert the data into production tables.

For a loading tutorial, see Use PolyBase to load data from Azure blob storage to Azure SQL Data Warehouse.

For more information, see Loading patterns blog.

Options for loading with PolyBase

PolyBase is a technology that accesses data outside of the database via the T-SQL language. It is the best way to load data into SQL Data Warehouse. With PolyBase, the data loads in parallel from the data source directly to the compute nodes.

To load data with PolyBase, you can use any of these loading options.

- PolyBase with T-SQL works well when your data is in Azure Blob storage or Azure Data Lake Store. It gives you the most control over the loading process, but also requires you to define external data objects. The other methods define these objects behind the scenes as you map source tables to destination tables. To orchestrate T-SQL loads, you can use Azure Data Factory, SSIS, or Azure functions.
- PolyBase with SSIS works well when your source data is in SQL Server, either SQL Server on-premises or in the cloud. SSIS defines the source to destination table mappings, and also orchestrates the load. If you already have SSIS packages, you can modify the packages to work with the new data warehouse destination.
- PolyBase with Azure Data Factory (ADF) is another orchestration tool. It defines a pipeline and schedules jobs.

PolyBase loads data from UTF-8 and UTF-16 encoded delimited text files. In addition to the delimited text files, it loads from the Hadoop file formats RC File, ORC, and Parquet. PolyBase can load data from Gzip and Snappy compressed files. PolyBase currently does not support extended ASCII, fixed-width format, and nested formats such as WinZip, JSON, and XML.

Non-PolyBase loading options

If your data is not compatible with PolyBase, you can use bcp or the SQLBulkCopy API. bcp loads directly to SQL Data Warehouse without going through Azure Blob storage, and is intended only for small loads. Note, the load performance of these options is significantly slower than PolyBase.

Extract source data

Getting data out of your source system depends on the source. The goal is to move the data into delimited text files. If you are using SQL Server, you can use bcp command-line tool to export the data.

Land data to Azure storage

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

These are 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.

For more information, see Moving data to and from Azure Storage

Prepare data

You might need to prepare and clean the data in your storage account before loading it into SQL Data Warehouse. 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.

Define external tables

Before you can load data, you need to define external tables in your data warehouse. PolyBase uses external tables to define and access the data in Azure Storage. The external table is similar to a regular table. The main difference is the external table points to data that is stored outside the data warehouse.

Defining external tables involves specifying the data source, the format of the text files, and the table definitions. These are the T-SQL syntax topics that you will need:

- CREATE EXTERNAL DATA SOURCE
- CREATE EXTERNAL FILE FORMAT
- CREATE EXTERNAL TABLE

For an example of creating external objects, see the Create external tables step in the loading tutorial.

Format text files

format definition. The data in each row of the text file must align with the table definition.

To format the text files:

- If your data is coming from a non-relational source, you need to transform it into rows and columns. Whether the data is from a relational or non-relational source, the data must be transformed to align with the column definitions for the table into which you plan to load the data.
- Format data in the text file to align with the columns and data types in the SQL Data Warehouse destination table. Misalignment between data types in the external text files and the data warehouse table causes rows to be rejected during the load.
- Separate fields in the text file with a terminator. Be sure to use a character or a character sequence that is not found in your source data. Use the terminator you specified with CREATE EXTERNAL FILE FORMAT.

Load to a staging table

To get data into the data warehouse, it works well to first load the data into a staging table. By using a staging table, you can handle errors without interfering with the production tables, and you avoid running rollback operations on the production table. A staging table also gives you the opportunity to use SQL Data Warehouse to run transformations before inserting the data into production tables.

To load with T-SQL, run the CREATE TABLE AS SELECT (CTAS) T-SQL statement. This command inserts the results of a select statement into a new table. When the statement selects from an external table, it imports the external data.

In the following example, ext.Date is an external table. All rows are imported into a new table called dbo.Date.

```
CREATE TABLE [dbo].[Date]
WITH
(
    CLUSTERED COLUMNSTORE INDEX
)
AS SELECT * FROM [ext].[Date]
;
```

Transform the data

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

Insert data into production table

The INSERT INTO ... SELECT statement moves the data from the staging table to the permanent table.

As you design an ETL process, try running the process on a small test sample. Try extracting 1000 rows from the table to a file, move it to Azure, and then try loading it into a staging table.

Partner loading solutions

Many of our partners have loading solutions. To find out more, see a list of our solution partners.

Next steps

For loading guidance, see Guidance for load data.

Best practices for loading data into Azure SQL Data Warehouse

1/18/2018 • 6 min to read • Edit Online

Recommendations and performance optimizations for loading data into Azure SQL Data Warehouse.

- To learn more about PolyBase and designing an Extract, Load, and Transform (ELT) process, see Design ELT for SQL Data Warehouse.
- For a loading tutorial, Use PolyBase to load data from Azure blob storage to Azure SQL Data Warehouse.

Preparing data in Azure Storage

To minimize latency, co-locate your storage layer and your data warehouse.

When exporting data into an ORC File Format, you might get Java out-of-memory errors when there are large text columns. To work around this limitation, export only a subset of the columns.

PolyBase cannot load rows that have more than 1,000,000 bytes of data. When you put data into the text files in Azure Blob storage or Azure Data Lake Store, they must have fewer than 1,000,000 bytes of data. This byte limitation is true regardless of the table schema.

All file formats have different performance characteristics. For the fastest load, use compressed delimited text files. The difference between UTF-8 and UTF-16 performance is minimal.

Split large compressed files into smaller compressed files.

Running loads with enough compute

For fastest loading speed, run only one load job at a time. If that is not feasible, run a minimal number of loads concurrently. If you expect a large loading job, consider scaling up your data warehouse before the load.

To run loads with appropriate compute resources, create loading users designated for running loads. Assign each loading user to a specific resource class. To run a load, log in as one of the loading users, and then run the load. The load runs with the user's resource class. This method is simpler than trying to change a user's resource class to fit the current resource class need.

Example of creating a loading user

This example creates a loading user for the staticrc20 resource class. The first step is to **connect to master** and create a login.

```
-- Connect to master
CREATE LOGIN LoaderRC20 WITH PASSWORD = 'a123STRONGpassword!';
```

Connect to the data warehouse and create a user. The following code assumes you are connected to the database called mySampleDataWarehouse. It shows how to create a user called LoaderRC20, give the user control permission on a database. It then adds the user as a member of the staticrc20 database role.

```
-- Connect to the database
CREATE USER LoaderRC20 FOR LOGIN LoaderRC20;
GRANT CONTROL ON DATABASE::[mySampleDataWarehouse] to LoaderRC20;
EXEC sp_addrolemember 'staticrc20', 'LoaderRC20';
```

To run a load with resources for the staticRC20 resource classes, simply log in as LoaderRC20 and run the load.

Run loads under static rather than dynamic resource classes. Using the static resource classes guarantees the same resources regardless of your service level. If you use a dynamic resource class, the resources vary according to your service level. For dynamic classes, a lower service level means you probably need to use a larger resource class for your loading user.

Allowing multiple users to load

There is often a need to have multiple users load data into a data warehouse. Loading with the [CREATE TABLE AS SELECT (Transact-SQL)][CREATE TABLE AS SELECT (Transact-SQL)] requires CONTROL permissions of the database. The CONTROL permission gives control access to all schemas. You might not want all loading users to have control access on all schemas. To limit permissions, use the DENY CONTROL statement.

For example, consider database schemas, schema_A for dept A, and schema_B for dept B. Let database users user_A and user_B be users for PolyBase loading in dept A and B, respectively. They both have been granted CONTROL database permissions. The creators of schema A and B now lock down their schemas using DENY:

```
DENY CONTROL ON SCHEMA :: schema_A TO user_B;
DENY CONTROL ON SCHEMA :: schema_B TO user_A;
```

User_A and user_B are now locked out from the other dept's schema.

Loading to a staging table

To achieve the fastest loading speed for moving data into a data warehouse table, load data into a staging table. Define the staging table as a heap and use round-robin for the distribution option.

Consider that loading is usually a two-step process in which you first load to a staging table and then insert the data into a production data warehouse table. If the production table uses a hash distribution, the total time to load and insert might be faster if you define the staging table with the hash distribution. Loading to the staging table takes longer, but the second step of inserting the rows to the production table does not incur data movement across the distributions.

Loading to a columnstore index

Columnstore indexes require large amounts of memory to compress data into high-quality rowgroups. For best compression and index efficiency, the columnstore index needs to compress the maximum of 1,048,576 rows into each rowgroup. When there is memory pressure, the columnstore index might not be able to achieve maximum compression rates. This in turn effects query performance. For a deep dive, see Columnstore memory optimizations.

- To ensure the loading user has enough memory to achieve maximum compression rates, use loading users that are a member of a medium or large resource class.
- Load enough rows to completely fill new rowgroups. During a bulk load, every 1,048,576 rows get compressed directly into the columnstore as a full rowgroup. Loads with fewer than 102,400 rows send the rows to the deltastore where rows are held in a b-tree index. If you load too few rows, they might all go to the deltastore and not get compressed immediately into columnstore format.

Handling loading failures

A load using an external table can fail with the error "*Query aborted-- the maximum reject threshold was reached while reading from an external source*". This message indicates that your external data contains dirty records. A data record is considered dirty if the data types and number of columns do not match the column definitions of the external table, or if the data doesn't conform to the specified external file format.

To fix the dirty records, ensure that your external table and external file format definitions are correct and your external data conforms to these definitions. In case a subset of external data records are dirty, you can choose to reject these records for your queries by using the reject options in CREATE EXTERNAL TABLE.

Inserting data into a production table

A one-time load to a small table with an INSERT statement, or even a periodic reload of a look-up might perform good enough with a statement like `INSERT INTO MyLookup VALUES (1, 'Type 1')`. However, singleton inserts are not as efficient as performing a bulk load.

If you have thousands or more single inserts throughout the day, batch the inserts so you can bulk load them. Develop your processes to append the single inserts to a file, and then create another process that periodically loads the file.

Creating statistics after the load

To improve query performance, it's important to create statistics on all columns of all tables after the first load, or substantial changes occur in the data. For a detailed explanation of statistics, see [Statistics][Statistics]. The following example creates statistics on five columns of the Customer_Speed table.

```
create statistics [SensorKey] on [Customer_Speed] ([SensorKey]);
create statistics [CustomerKey] on [Customer_Speed] ([CustomerKey]);
create statistics [GeographyKey] on [Customer_Speed] ([GeographyKey]);
create statistics [Speed] on [Customer_Speed] ([Speed]);
create statistics [YearMeasured] on [Customer_Speed] ([YearMeasured]);
```

Rotate storage keys

It is good security practice to change the access key to your blob storage on a regular basis. You have two storage keys for your blob storage account, which enables you to transition the keys.

To rotate Azure Storage account keys:

1. Create a second database scoped credential based on the secondary storage access key.
2. Create a second external data source based off this new credential.
3. Drop and create the external table(s) so they point to the new external data sources.

After migrating your external tables to the new data source, perform the following clean-up tasks:

1. Drop the first external data source.
2. Drop the first database scoped credential based on the primary storage access key.
3. Log in to Azure and regenerate the primary access key so it is ready for your next rotation.

Next steps

To monitor data loads, see Monitor your workload using DMVs.

Leverage other services with SQL Data Warehouse

6/27/2017 • 1 min to read • Edit Online

In addition to its core functionality, SQL Data Warehouse enables users to leverage many of the other services in Azure alongside it. Specifically, we have currently taken steps to deeply integrate with the following:

- Power BI
- Azure Data Factory
- Azure Machine Learning
- Azure Stream Analytics

We are working to connect with more services across the Azure ecosystem.

Power BI

Power BI integration allows you to leverage the compute power of SQL Data Warehouse with the dynamic reporting and visualization of Power BI. Power BI integration currently includes:

- **Direct Connect**: A more advanced connection with logical pushdown against SQL Data Warehouse. This provides faster analysis on a larger scale.
- **Open in Power BI**: The 'Open in Power BI' button passes instance information to Power BI, allowing for a more seamless connection.

See Integrate with Power BI or the Power BI documentation for more information.

Azure Data Factory

Azure Data Factory gives users a managed platform to create complex Extract-Load pipelines. SQL Data Warehouse's integration with Azure Data Factory includes the following:

- **Stored Procedures**: Orchestrate the execution of stored procedures on SQL Data Warehouse.
- **Copy**: Use ADF to move data into SQL Data Warehouse. This operation can use ADF's standard data movement mechanism or PolyBase under the covers.

See Integrate with Azure Data Factory or the Azure Data Factory documentation for more information.

Azure Machine Learning

Azure Machine Learning is a fully managed analytics service which allows users to create intricate models leveraging a large set of predictive tools. SQL Data Warehouse is supported as both a source and destination for these models with the following functionality:

- **Read Data:** Drive models at scale using T-SQL against SQL Data Warehouse.
- **Write Data:** Commit changes from any model back to SQL Data Warehouse.

See Integrate with Azure Machine Learning or the Azure Machine Learning documentation for more information.

Azure Stream Analytics

Azure Stream Analytics is a complex, fully managed infrastructure for processing and consuming event data generated from Azure Event Hub. Integration with SQL Data Warehouse allows for streaming data to be effectively

See Integrate with Azure Stream Analytics or the Azure Stream Analytics documentation for more information.

How to use Elastic Query with SQL Data Warehouse

12/8/2017 • 7 min to read • Edit Online

Elastic Query with Azure SQL Data Warehouse allows you to write Transact-SQL in a SQL Database that is sent remotely to an Azure SQL Data Warehouse instance through the use of external tables. Using this feature provides cost-savings and more performant architectures depending on the scenario.

This feature enables two primary scenarios:

1. Domain isolation
2. Remote query execution

Domain isolation

Domain isolation refers to the classic data mart scenario. In certain scenarios, one may want to provide a logical domain of data to downstream users that are isolated from the rest of the data warehouse, for a variety of reasons including but not limited to:

1. Resource Isolation - SQL database is optimized to serve a large base of concurrent users serving slightly different workloads than the large analytical queries that the data warehouse is reserved for. Isolation ensures the right workloads are served by the right tools.
2. Security isolation - to separate an authorized data subset selectively via certain schemas.
3. Sandboxing - provide a sample set of data as a "playground" to explore production queries etc.

Elastic query can provide the ability to easily select subsets of SQL data warehouse data and move it into a SQL database instance. Furthermore this isolation does not preclude the ability to also enable Remote query execution allowing for more interesting "cache" scenarios.

Remote query execution

Elastic query allows for remote query execution on a SQL data warehouse instance. One can utilize the best of both SQL database and SQL data warehouse by separating your hot and cold data between the two databases. Users can keep more recent data within a SQL database, which can serve reports and large numbers of average business users. However, when more data or computation is needed, a user can offload part of the query to a SQL data warehouse instance where large-scale aggregates can be processed much faster and more efficiently.

Elastic Query overview

An elastic query can be used to make data located within a SQL data warehouse available to SQL database instances. Elastic query allows queries from a SQL database refer to tables in a remote SQL data warehouse instance.

The first step is to create an external data source definition that refers to the SQL data warehouse instance, which uses existing user credentials within the SQL data warehouse. No changes are necessary on the remote SQL data warehouse instance.

> **IMPORTANT**
> You must possess ALTER ANY EXTERNAL DATA SOURCE permission. This permission is included with the ALTER DATABASE permission. ALTER ANY EXTERNAL DATA SOURCE permissions are needed to refer to remote data sources.

Next we create a remote external table definition in a SQL database instance which points to a remote table in the

set is sent back to the calling SQL database instance. For a brief tutorial of setting up an Elastic Query between SQL database and SQL data warehouse, see the Configure Elastic Query with SQL Data Warehouse.

For more information on Elastic Query with SQL database, see the Azure SQL Database elastic query overview.

Best practices

General

- When using remote query execution, ensure you're only selecting necessary columns and applying the right filters. Not only does this increase the compute necessary, but it also increases the size of the result set and therefore the amount of data that need to be moved between the two instances.
- Maintain data for analytical purposes in both SQL Data Warehouse and SQL Database in clustered columnstore for analytilcal performance.
- Ensure that source tables are partitioned for query and data movement.
- Ensure SQL database instances used as a cache are partitioned to enable more granular updates and easier management.
- Ideally use PremiumRS databases because they provide the analytical benefits of clustered columnstore indexing with a focus on IO-intensive workloads at a discount from Premium databases.
- After loads, utilize load or effective date identification columns for upserts in the SQL Database instances to maintain cache-source integrity.
- Create a separate login and user in your SQL data warehouse instance for your SQL database remote login credentials defined in the external data source.

Elastic Querying

- In many cases, one might want to manage a type of stretched table, where a portion of your table is within the SQL Database as cached data for performance with the rest of the data stored in SQL Data Warehouse. You will need to have two objects in SQL Database: an external table within SQL Database that references the base table in SQL Data Warehouse, and the "cached" portion of the table within the SQL Database. Consider creating a view over the top of the cached portion of the table and the external table which unions both tables and applies filters that separate data materialized within SQL Database and SQL Data Warehouse data exposed through external tables.

  Imagine we would like to keep the most recent year of data in a SQL database instance. We have two tables **ext.Orders**, which references the data warehouse orders tables, and **dbo.Orders** which represents the most recent years worth of data within the SQL database instance. Instead of asking users to decide whether to query one table or another, we create a view over the top of both tables on the partition point of the most recent year.

```
CREATE VIEW dbo.Orders_Elastic AS
SELECT
  [col_a]
, [col_b]
, ...
, [col_n]
FROM
  [dbo].[Orders]
WHERE
  YEAR([o_orderdate]) >= '<Most Recent Year>'
UNION
SELECT
  [col_a]
, [col_b]
, ...
, [col_n]
FROM
  [ext].[Orders]
WHERE
  YEAR([o_orderdate]) < '<Most Recent Year>'
```

A view produced in such a way let's the query compiler determine whether it needs to use the data warehouse instance to answer your users query.

There is overhead of submitting, compiling, running, and moving data associated with each elastic query against the data warehouse instance. Be cognizant that each elastic query counts against your concurrency slots and uses resources.

- If one plans to drill down further into the result set from the data warehouse instance, consider materializing it in a temp table in the SQL Database for performance and to prevent unnecessary resource usage.

Moving data

- If possible, keep data management easier with append-only source tables such that updates are easily maintainable between the data warehouse and database instances.
- Move data on the partition level with flush and fill semantics to minimize the query cost on the data warehouse level and the amount of data moved to keep the database instance up to date.

When to choose Azure Analysis Services vs SQL Database

Azure Analysis Services

- You plan on using your cache with a BI tool that submits large numbers of small queries
- You need subsecond query latency
- You are experienced in managing/developing models for Analysis Services

SQL Database

- You want to query your cache data with SQL
- You need remote execution for certain queries
- You have larger cache requirements

FAQ

Q: Can I use databases within an Elastic Pool with Elastic Query?

A: Yes. SQL Databases within an Elastic Pool can use Elastic Query.

Q: Is there a cap for how many databases I can use for Elastic Query?

A: There is no hard cap on how many databases can be used for Elastic Query. However, each Elastic Query (queries that hit SQL Data Warehouse) will count toward normal concurrency limits.

A: DTU limits are not imposed any differently with Elastic Query. The standard policy is such that logical servers have DTU limits in place to prevent customers from accidental overspending. If you are enabling several databases for elastic query alongside a SQL Data Warehouse instance, you may hit the cap unexpectedly. If this occurs, submit a request to increase DTU limit on your logical server. You can increase your quota by creating a support ticket and selecting *Quota* as the request type

Q: Can I use row level security/Dynamic Data Masking with Elastic Query?

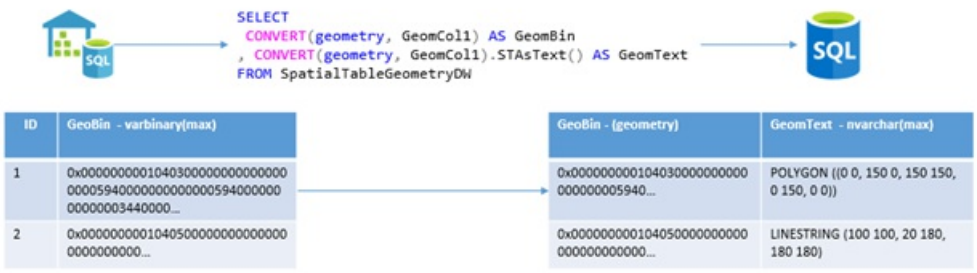
A: Customers who wish to use more advanced security features with SQL Database can do so by first moving and storing the data in the SQL Database. You cannot currently apply row level security or DDM on data queried through external tables.

Q: Can I write from my SQL database instance to data warehouse instance?

A: Currently this feature is not supported. Visit our Feedback page to create/vote for this functionality if this is a feature you would like to see in the future.

Q: Can I use spatial types like geometry/geography?

A: You can store spatial types in SQL Data Warehouse as varbinary(max) values. When you query these columns using elastic query, you can convert them to the appropriate types at runtime.



Resource classes for workload management

10/24/2017 • 14 min to read • Edit Online

Guidance for using resource classes to manage the number of concurrent queries that run concurrently, and compute resources for queries in Azure SQL Data Warehouse.

What is workload management?

Workload management is the ability to optimize the overall performance of all queries. A well-tuned workload runs queries and load operations efficiently regardless of whether they are compute-intensive or IO-intensive.

SQL Data Warehouse provides workload management capabilities for multi-user environments. A data warehouse is not intended for multi-tenant workloads.

What are resource classes?

Resource classes are pre-determined resource limits that govern query execution. SQL Data Warehouse limits the compute resources for each query according to resource class.

Resource classes help you manage the overall performance of your data warehouse workload.Using resource classes effectively helps you manage your workload by setting limits on the number of queries that run concurrently and the compute-resources assigned to each query.

- Smaller resource classes use less compute resources but enable greater overall query concurrency
- Larger resource classes provide more compute resources but restrict the query concurrency

Resource classes are designed for data management and manipulation activities. Some very complex queries will also benefit when there are large joins and sorts so that the system executes the query in memory rather than spilling to disk.

The following operations are governed by resource classes:

- INSERT-SELECT, UPDATE, DELETE
- SELECT (when querying user tables)
- ALTER INDEX - REBUILD or REORGANIZE
- ALTER TABLE REBUILD
- CREATE INDEX
- CREATE CLUSTERED COLUMNSTORE INDEX
- CREATE TABLE AS SELECT (CTAS)
- Data loading
- Data movement operations conducted by the Data Movement Service (DMS)

> **NOTE**
> SELECT statements on dynamic management views (DMVs) or other system views are not governed by any of the concurrency limits. You can monitor the system regardless of the number of queries executing on it.

Static and dynamic resource classes

There are two types of resource classes: dynamic and static

which is measured in data warehouse units. This static allocation means on larger service levels you can run more queries in each resource class. The static resource classes are named staticrc10, staticrc20, staticrc30, staticrc40, staticrc50, staticrc60, staticrc70, and staticrc80. These resource classes are best suited to solutions which increase resource class to get additional compute resources.

- **Dynamic Resource Classes** allocate a variable amount of memory depending on the current service level. When you scale up to a larger service level, your queries automatically get more memory. The dynamic resource classes are named smallrc, mediumrc, largerc, and xlargerc. These resource classes are best suited to solutions which increase compute scale to get additional resources.

The performance tiers use the same resource class names, but have different memory and concurrency specifications.

Assigning resource classes

Resource classes are implemented by assigning users to database roles. When a user runs a query, the query runs with the user's resource class. For example, when a user is a member of the smallrc or staticrc10 database role, their queries run with small amounts of memory. When a database user is a member of the xlargerc or staticrc80 database roles, their queries run with large amounts of memory.

To increase a user's resource class, use the stored procedure sp_addrolemember.

```
EXEC sp_addrolemember 'largerc', 'loaduser';
```

To decrease the resource class, use sp_droprolemember.

```
EXEC sp_droprolemember 'largerc', 'loaduser';
```

The resource class of the service administrator is fixed and cannot be changed. The service administrator is the user created during the provisioning process.

Default resource class

By default, each user is a member of the small resource class, **smallrc**.

Resource class precedence

Users can be members of multiple resource classes. When a user belongs to more than one resource class:

- Dynamic resource classes take precedence over static resource classes. For example, if a user is a member of both mediumrc(dynamic) and staticrc80 (static), queries run with mediumrc.
- Larger resource classes take precedence over smaller resource classes. For example, if a user is a member of mediumrc and largerc, queries run with largerc. Likewise, if a user is a member of both staticrc20 and statirc80, queries run with staticrc80 resource allocations.

Queries exempt from resource classes

Some queries always run in the smallrc resource class even though the user is a member of a larger resource class. These exempt queries do not count towards the concurrency limit. For example, if the concurrency limit is 16, many users can be selecting from system views without impacting the available concurrency slots.

The following statements are exempt from resource classes and always run in smallrc:

- CREATE or DROP TABLE
- ALTER TABLE ... SWITCH, SPLIT, or MERGE PARTITION
- ALTER INDEX DISABLE

- CREATE, UPDATE, or DROP STATISTICS
- TRUNCATE TABLE
- ALTER AUTHORIZATION
- CREATE LOGIN
- CREATE, ALTER, or DROP USER
- CREATE, ALTER, or DROP PROCEDURE
- CREATE or DROP VIEW
- INSERT VALUES
- SELECT from system views and DMVs
- EXPLAIN
- DBCC

Recommendations

We recommend creating a user that is dedicated to running a specific type of query or load operations. Then give that user a permanent resource class instead of changing the resource class on a frequent basis. Given that static resource classes afford greater overall control on the workload we also suggest using those first before considering dynamic resource classes.

Resource classes for load users

`CREATE TABLE` uses clustered columnstore indexes by default. Compressing data into a columnstore index is a memory-intensive operation, and memory pressure can reduce the index quality. Therefore, you are most likely to require a higher resource class when loading data. To ensure loads have enough memory, you can create a user that is designated for running loads and assign that user to a higher resource class.

The memory needed to process loads efficiently depends on the nature of the table loaded and the data size. For more information on memory requirements, see Maximizing rowgroup quality.

Once you have determined the memory requirement, choose whether to assign the load user to a static or dynamic resource class.

- Use a static resource class when table memory requirements fall within a specific range. Loads run with appropriate memory. When you scale the data warehouse, the loads do not need more memory. By using a static resource class, the memory allocations stay constant. This consistency conserves memory and allows more queries to run concurrently. We recommend that new solutions use the static resource classes first as these provide greater control.
- Use a dynamic resource class when table memory requirements vary widely. Loads might require more memory than the current DWU or cDWU level provides. Therefore, scaling the data warehouse adds more memory to load operations, which allows loads to perform faster.

Resource classes for queries

Some queries are compute-intensive and some are not.

- Choose a dynamic resource class when queries are complex, but do not need high concurrency. For example, generating daily or weekly reports is an occasional need for resources. If the reports are processing large amounts of data, scaling the data warehouse provides more memory to the user's existing resource class.
- Choose a static resource class when resource expectations vary throughout the day. For example, a static resource class works well when the data warehouse is queried by many people. When scaling the data warehouse, the amount of memory allocated to the user does not change. Consequently, more queries can be executed in parallel on the system.

Selecting a proper memory grant depends on many factors, such as the amount of data queried, the nature of

queries to complete faster, but reduces the overall concurrency. If concurrency is not an issue, over-allocating memory does not harm throughput.

To tune performance, use different resource classes. The next section gives a stored procedure that helps you figure out the best resource class.

Example code for finding the best resource class

You can use the following stored procedure to figure out concurrency and memory grant per resource class at a given SLO and the closest best resource class for memory intensive CCI operations on non-partitioned CCI table at a given resource class:

Here's the purpose of this stored procedure:

1. To see the concurrency and memory grant per resource class at a given SLO. User needs to provide NULL for both schema and tablename as shown in this example.
2. To see the closest best resource class for the memory-intensive CCI operations (load, copy table, rebuild index, etc.) on non partitioned CCI table at a given resource class. The stored proc uses table schema to find out the required memory grant.

Dependencies & Restrictions:

- This stored procedure is not designed to calculate the memory requirement for a partitioned cci table.
- This stored procedure doesn't take memory requirement into account for the SELECT part of CTAS/INSERT-SELECT and assumes it is a SELECT.
- This stored procedure uses a temp table, which is available in the session where this stored procedure was created.
- This stored procedure depends on the current offerings (for example, hardware configuration, DMS config), and if any of that changes then this stored proc does not work correctly.
- This stored procedure depends on existing offered concurrency limit and if that changes then this stored procedure would not work correctly.
- This stored procedure depends on existing resource class offerings and if that changes then this stored procedure would not work correctly.

> **NOTE**
>
> If you are not getting output after executing stored procedure with parameters provided, then there could be two cases.
> 1. Either DW Parameter contains an invalid SLO value
> 2. Or, there is no matching resource class for the CCI operation on the table.
> For example, at DW100, the highest memory grant available is 400 MB, and if table schema is wide enough to cross the requirement of 400 MB.

Usage example:

Syntax:

```
EXEC dbo.prc_workload_management_by_DWU @DWU VARCHAR(7), @SCHEMA_NAME VARCHAR(128), @TABLE_NAME VARCHAR(128)
```

1. @DWU: Either provide a NULL parameter to extract the current DWU from the DW DB or provide any supported DWU in the form of 'DW100'
2. @SCHEMA_NAME: Provide a schema name of the table
3. @TABLE_NAME: Provide a table name of the interest

Examples executing this stored proc:

```
EXEC dbo.prc_workload_management_by_DWU 'DW2000', 'dbo', 'Table1';
EXEC dbo.prc_workload_management_by_DWU NULL, 'dbo', 'Table1';
EXEC dbo.prc_workload_management_by_DWU 'DW6000', NULL, NULL;
EXEC dbo.prc_workload_management_by_DWU NULL, NULL, NULL;
```

The following statement creates Table1 that is used in the preceding examples.

```
CREATE TABLE Table1 (a int, b varchar(50), c decimal (18,10), d char(10), e varbinary(15), f float, g
datetime, h date);
```

Stored procedure definition

```
--------------------------------------------------------------------------------
-- Dropping prc_workload_management_by_DWU procedure if it exists.
--------------------------------------------------------------------------------
IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P' AND name = 'prc_workload_management_by_DWU')
DROP PROCEDURE dbo.prc_workload_management_by_DWU
GO

--------------------------------------------------------------------------------
-- Creating prc_workload_management_by_DWU.
--------------------------------------------------------------------------------
CREATE PROCEDURE dbo.prc_workload_management_by_DWU
(@DWU VARCHAR(7),
 @SCHEMA_NAME VARCHAR(128),
 @TABLE_NAME VARCHAR(128)
)
AS
IF @DWU IS NULL
BEGIN
-- Selecting proper DWU for the current DB if not specified.
SET @DWU = (
  SELECT 'DW'+CAST(COUNT(*)*100 AS VARCHAR(10))
  FROM sys.dm_pdw_nodes
  WHERE type = 'COMPUTE')
END

DECLARE @DWU_NUM INT
SET @DWU_NUM = CAST (SUBSTRING(@DWU, 3, LEN(@DWU)-2) AS INT)

-- Raise error if either schema name or table name is supplied but not both them supplied
--IF ((@SCHEMA_NAME IS NOT NULL AND @TABLE_NAME IS NULL) OR (@TABLE_NAME IS NULL AND @SCHEMA_NAME IS NOT
NULL))
--     RAISEERROR('User need to supply either both Schema Name and Table Name or none of them')

-- Dropping temp table if exists.
IF OBJECT_ID('tempdb..#ref') IS NOT NULL
BEGIN
  DROP TABLE #ref;
END

-- Creating ref. temp table (CTAS) to hold mapping info.
-- CREATE TABLE #ref
CREATE TABLE #ref
WITH (DISTRIBUTION = ROUND_ROBIN)
AS
WITH
-- Creating concurrency slots mapping for various DWUs.
alloc
AS
(
  SELECT 'DW100' AS DWU, 4 AS max_queries, 4 AS max_slots, 1 AS slots_used_smallrc, 1 AS
slots_used_mediumrc,
        2 AS slots_used_largerc, 4 AS slots_used_xlargerc, 1 AS slots_used_staticrc10, 2 AS
slots_used_staticrc20,
        4 AS slots_used_staticrc30, 4 AS slots_used_staticrc40, 4 AS slots_used_staticrc50,
```

```
    SELECT 'DW200', 8, 8, 1, 2, 4, 8, 1, 2, 4, 8, 8, 8, 8, 8
  UNION ALL
    SELECT 'DW300', 12, 12, 1, 2, 4, 8, 1, 2, 4, 8, 8, 8, 8, 8
  UNION ALL
    SELECT 'DW400', 16, 16, 1, 4, 8, 16, 1, 2, 4, 8, 16, 16, 16, 16
  UNION ALL
     SELECT 'DW500', 20, 20, 1, 4, 8, 16, 1, 2, 4, 8, 16, 16, 16, 16
  UNION ALL
    SELECT 'DW600', 24, 24, 1, 4, 8, 16, 1, 2, 4, 8, 16, 16, 16, 16
  UNION ALL
    SELECT 'DW1000', 32, 40, 1, 8, 16, 32, 1, 2, 4, 8, 16, 32, 32, 32
  UNION ALL
    SELECT 'DW1200', 32, 48, 1, 8, 16, 32, 1, 2, 4, 8, 16, 32, 32, 32
  UNION ALL
    SELECT 'DW1500', 32, 60, 1, 8, 16, 32, 1, 2, 4, 8, 16, 32, 32, 32
  UNION ALL
    SELECT 'DW2000', 32, 80, 1, 16, 32, 64, 1, 2, 4, 8, 16, 32, 64, 64
  UNION ALL
   SELECT 'DW3000', 32, 120, 1, 16, 32, 64, 1, 2, 4, 8, 16, 32, 64, 64
  UNION ALL
    SELECT 'DW6000', 32, 240, 1, 32, 64, 128, 1, 2, 4, 8, 16, 32, 64, 128
)
-- Creating workload mapping to their corresponding slot consumption and default memory grant.
,map
AS
(
  SELECT 'SloDWGroupC00' AS wg_name,1 AS slots_used,100 AS tgt_mem_grant_MB
  UNION ALL
    SELECT 'SloDWGroupC01',2,200
  UNION ALL
    SELECT 'SloDWGroupC02',4,400
  UNION ALL
    SELECT 'SloDWGroupC03',8,800
  UNION ALL
    SELECT 'SloDWGroupC04',16,1600
  UNION ALL
    SELECT 'SloDWGroupC05',32,3200
  UNION ALL
    SELECT 'SloDWGroupC06',64,6400
  UNION ALL
    SELECT 'SloDWGroupC07',128,12800
)
-- Creating ref based on current / asked DWU.
, ref
AS
(
  SELECT  a1.*
  ,       m1.wg_name          AS wg_name_smallrc
  ,       m1.tgt_mem_grant_MB AS tgt_mem_grant_MB_smallrc
  ,       m2.wg_name          AS wg_name_mediumrc
  ,       m2.tgt_mem_grant_MB AS tgt_mem_grant_MB_mediumrc
  ,       m3.wg_name          AS wg_name_largerc
  ,       m3.tgt_mem_grant_MB AS tgt_mem_grant_MB_largerc
  ,       m4.wg_name          AS wg_name_xlargerc
  ,       m4.tgt_mem_grant_MB AS tgt_mem_grant_MB_xlargerc
  ,       m5.wg_name          AS wg_name_staticrc10
  ,       m5.tgt_mem_grant_MB AS tgt_mem_grant_MB_staticrc10
  ,       m6.wg_name          AS wg_name_staticrc20
  ,       m6.tgt_mem_grant_MB AS tgt_mem_grant_MB_staticrc20
  ,       m7.wg_name          AS wg_name_staticrc30
  ,       m7.tgt_mem_grant_MB AS tgt_mem_grant_MB_staticrc30
  ,       m8.wg_name          AS wg_name_staticrc40
  ,       m8.tgt_mem_grant_MB AS tgt_mem_grant_MB_staticrc40
  ,       m9.wg_name          AS wg_name_staticrc50
  ,       m9.tgt_mem_grant_MB AS tgt_mem_grant_MB_staticrc50
  ,       m10.wg_name          AS wg_name_staticrc60
  ,       m10.tgt_mem_grant_MB AS tgt_mem_grant_MB_staticrc60
```

```
  ,       m12.wg_name          AS wg_name_staticrc80
  ,       m12.tgt_mem_grant_MB AS tgt_mem_grant_MB_staticrc80
  FROM alloc a1
  JOIN map   m1  ON a1.slots_used_smallrc     = m1.slots_used
  JOIN map   m2  ON a1.slots_used_mediumrc    = m2.slots_used
  JOIN map   m3  ON a1.slots_used_largerc     = m3.slots_used
  JOIN map   m4  ON a1.slots_used_xlargerc    = m4.slots_used
  JOIN map   m5  ON a1.slots_used_staticrc10    = m5.slots_used
  JOIN map   m6  ON a1.slots_used_staticrc20    = m6.slots_used
  JOIN map   m7  ON a1.slots_used_staticrc30    = m7.slots_used
  JOIN map   m8  ON a1.slots_used_staticrc40    = m8.slots_used
  JOIN map   m9  ON a1.slots_used_staticrc50    = m9.slots_used
  JOIN map   m10  ON a1.slots_used_staticrc60    = m10.slots_used
  JOIN map   m11  ON a1.slots_used_staticrc70    = m11.slots_used
  JOIN map   m12  ON a1.slots_used_staticrc80    = m12.slots_used
-- WHERE   a1.DWU = @DWU
  WHERE   a1.DWU = UPPER(@DWU)
)
SELECT  DWU
,       max_queries
,       max_slots
,       slots_used
,       wg_name
,       tgt_mem_grant_MB
,       up1 as rc
,       (ROW_NUMBER() OVER(PARTITION BY DWU ORDER BY DWU)) as rc_id
FROM
(
    SELECT  DWU
    ,       max_queries
    ,       max_slots
    ,       slots_used
    ,       wg_name
    ,       tgt_mem_grant_MB
    ,       REVERSE(SUBSTRING(REVERSE(wg_names),1,CHARINDEX('_',REVERSE(wg_names),1)-1)) as up1
    ,       REVERSE(SUBSTRING(REVERSE(tgt_mem_grant_MBs),1,CHARINDEX('_',REVERSE(tgt_mem_grant_MBs),1)-1))
as up2
    ,       REVERSE(SUBSTRING(REVERSE(slots_used_all),1,CHARINDEX('_',REVERSE(slots_used_all),1)-1)) as up3
    FROM    ref AS r1
    UNPIVOT
    (
        wg_name FOR wg_names IN (wg_name_smallrc,wg_name_mediumrc,wg_name_largerc,wg_name_xlargerc,
        wg_name_staticrc10, wg_name_staticrc20, wg_name_staticrc30, wg_name_staticrc40, wg_name_staticrc50,
        wg_name_staticrc60, wg_name_staticrc70, wg_name_staticrc80)
    ) AS r2
    UNPIVOT
    (
        tgt_mem_grant_MB FOR tgt_mem_grant_MBs IN (tgt_mem_grant_MB_smallrc,tgt_mem_grant_MB_mediumrc,
        tgt_mem_grant_MB_largerc,tgt_mem_grant_MB_xlargerc, tgt_mem_grant_MB_staticrc10,
tgt_mem_grant_MB_staticrc20,
        tgt_mem_grant_MB_staticrc30, tgt_mem_grant_MB_staticrc40, tgt_mem_grant_MB_staticrc50,
        tgt_mem_grant_MB_staticrc60, tgt_mem_grant_MB_staticrc70, tgt_mem_grant_MB_staticrc80)
    ) AS r3
    UNPIVOT
    (
        slots_used FOR slots_used_all IN (slots_used_smallrc,slots_used_mediumrc,slots_used_largerc,
        slots_used_xlargerc, slots_used_staticrc10, slots_used_staticrc20, slots_used_staticrc30,
        slots_used_staticrc40, slots_used_staticrc50, slots_used_staticrc60, slots_used_staticrc70,
        slots_used_staticrc80)
    ) AS r4
) a
WHERE   up1 = up2
AND     up1 = up3
;
-- Getting current info about workload groups.
WITH
dmv
```

```
  SELECT
          rp.name                                             AS rp_name
  ,       rp.max_memory_kb*1.0/1048576                        AS rp_max_mem_GB
  ,       (rp.max_memory_kb*1.0/1024)
          *(request_max_memory_grant_percent/100)             AS max_memory_grant_MB
  ,       (rp.max_memory_kb*1.0/1048576)
          *(request_max_memory_grant_percent/100)             AS max_memory_grant_GB
  ,       wg.name                                             AS wg_name
  ,       wg.importance                                       AS importance
  ,       wg.request_max_memory_grant_percent                 AS request_max_memory_grant_percent
  FROM    sys.dm_pdw_nodes_resource_governor_workload_groups wg
  JOIN    sys.dm_pdw_nodes_resource_governor_resource_pools rp    ON  wg.pdw_node_id  = rp.pdw_node_id
                                                                  AND wg.pool_id      = rp.pool_id
  WHERE   rp.name = 'SloDWPool'
  GROUP BY
          rp.name
  ,       rp.max_memory_kb
  ,       wg.name
  ,       wg.importance
  ,       wg.request_max_memory_grant_percent
)
-- Creating resource class name mapping.
,names
AS
(
  SELECT 'smallrc' as resource_class, 1 as rc_id
  UNION ALL
    SELECT 'mediumrc', 2
  UNION ALL
    SELECT 'largerc', 3
  UNION ALL
    SELECT 'xlargerc', 4
  UNION ALL
    SELECT 'staticrc10', 5
  UNION ALL
    SELECT 'staticrc20', 6
  UNION ALL
    SELECT 'staticrc30', 7
  UNION ALL
    SELECT 'staticrc40', 8
  UNION ALL
    SELECT 'staticrc50', 9
  UNION ALL
    SELECT 'staticrc60', 10
  UNION ALL
    SELECT 'staticrc70', 11
  UNION ALL
    SELECT 'staticrc80', 12
)
,base AS
(   SELECT  schema_name
    ,       table_name
    ,       SUM(column_count)                   AS column_count
    ,       ISNULL(SUM(short_string_column_count),0)   AS short_string_column_count
    ,       ISNULL(SUM(long_string_column_count),0)    AS long_string_column_count
    FROM    (   SELECT  sm.name                                             AS schema_name
                ,       tb.name                                             AS table_name
                ,       COUNT(co.column_id)                                 AS column_count
                           ,       CASE    WHEN co.system_type_id IN (36,43,106,108,165,167,173,175,231,239)
                                   AND  co.max_length <= 32
                                   THEN COUNT(co.column_id)
                           END                                              AS short_string_column_count
                ,       CASE    WHEN co.system_type_id IN (165,167,173,175,231,239)
                                   AND  co.max_length > 32 and co.max_length <=8000
                                   THEN COUNT(co.column_id)
                           END                                              AS long_string_column_count
                FROM    sys.schemas AS sm
```

```
                          WHERE tb.name = @TABLE_NAME AND sm.name = @SCHEMA_NAME
               GROUP BY sm.name
               ,        tb.name
               ,        co.system_type_id
               ,        co.max_length            ) a
GROUP BY schema_name
,        table_name
)
, size AS
(
SELECT  schema_name
,       table_name
,       75497472                                            AS table_overhead

,       column_count*1048576*8                              AS column_size
,       short_string_column_count*1048576*32                       AS short_string_size,
(long_string_column_count*16777216) AS long_string_size
FROM    base
UNION
SELECT CASE WHEN COUNT(*) = 0 THEN 'EMPTY' END as schema_name
         ,CASE WHEN COUNT(*) = 0 THEN 'EMPTY' END as table_name
         ,CASE WHEN COUNT(*) = 0 THEN 0 END as table_overhead
         ,CASE WHEN COUNT(*) = 0 THEN 0 END as column_size
         ,CASE WHEN COUNT(*) = 0 THEN 0 END as short_string_size

,CASE WHEN COUNT(*) = 0 THEN 0 END as long_string_size
FROM   base
)
, load_multiplier as
(
SELECT  CASE
                     WHEN FLOOR(8 * (CAST (@DWU_NUM AS FLOAT)/6000)) > 0 THEN FLOOR(8 * (CAST (@DWU_NUM AS
FLOAT)/6000))
                     ELSE 1
              END AS multipliplication_factor
)
       SELECT  r1.DWU
       , schema_name
       , table_name
       , rc.resource_class as closest_rc_in_increasing_order
       , max_queries_at_this_rc = CASE
            WHEN (r1.max_slots / r1.slots_used > r1.max_queries)
                 THEN r1.max_queries
            ELSE r1.max_slots / r1.slots_used
                 END
       , r1.max_slots as max_concurrency_slots
       , r1.slots_used as required_slots_for_the_rc
       , r1.tgt_mem_grant_MB  as rc_mem_grant_MB
       ,
CAST((table_overhead*1.0+column_size+short_string_size+long_string_size)*multipliplication_factor/1048576
AS DECIMAL(18,2)) AS est_mem_grant_required_for_cci_operation_MB
       FROM    size, load_multiplier, #ref r1, names  rc
       WHERE r1.rc_id=rc.rc_id
                     AND
CAST((table_overhead*1.0+column_size+short_string_size+long_string_size)*multipliplication_factor/1048576
AS DECIMAL(18,2)) < r1.tgt_mem_grant_MB
       ORDER BY
ABS(CAST((table_overhead*1.0+column_size+short_string_size+long_string_size)*multipliplication_factor/104857
6    AS DECIMAL(18,2)) - r1.tgt_mem_grant_MB)
GO
```

Next steps

For more information about managing database users and security, see Secure a database in SQL Data

Maximizing rowgroup quality for columnstore

10/24/2017 • 6 min to read • Edit Online

Rowgroup quality is determined by the number of rows in a rowgroup. Reduce memory requirements or increase the available memory to maximize the number of rows a columnstore index compresses into each rowgroup. Use these methods to improve compression rates and query performance for columnstore indexes.

Why the rowgroup size matters

Since a columnstore index scans a table by scanning column segments of individual rowgroups, maximizing the number of rows in each rowgroup enhances query performance. When rowgroups have a high number of rows, data compression improves which means there is less data to read from disk.

For more information about rowgroups, see Columnstore Indexes Guide.

Target size for rowgroups

For best query performance, the goal is to maximize the number of rows per rowgroup in a columnstore index. A rowgroup can have a maximum of 1,048,576 rows. It's okay to not have the maximum number of rows per rowgroup. Columnstore indexes achieve good performance when rowgroups have at least 100,000 rows.

Rowgroups can get trimmed during compression

During a bulk load or columnstore index rebuild, sometimes there isn't enough memory available to compress all the rows designated for each rowgroup. When there is memory pressure, columnstore indexes trim the rowgroup sizes so compression into the columnstore can succeed.

When there is insufficient memory to compress at least 10,000 rows into each rowgroup, SQL Data Warehouse generates an error.

For more information on bulk loading, see Bulk load into a clustered columnstore index.

How to monitor rowgroup quality

There is a DMV (sys.dm_pdw_nodes_db_column_store_row_group_physical_stats) that exposes useful information such as number of rows in rowgroups and the reason for trimming if there was trimming. You can create the following view as a handy way to query this DMV to get information on rowgroup trimming.

```
create view dbo.vCS_rg_physical_stats
as
with cte
as
(
select   tb.[name]                    AS [logical_table_name]
,        rg.[row_group_id]            AS [row_group_id]
,        rg.[state]                   AS [state]
,        rg.[state_desc]              AS [state_desc]
,        rg.[total_rows]              AS [total_rows]
,        rg.[trim_reason_desc]        AS trim_reason_desc
,        mp.[physical_name]           AS physical_name
FROM    sys.[schemas] sm
JOIN    sys.[tables] tb               ON  sm.[schema_id]          = tb.[schema_id]
JOIN    sys.[pdw_table_mappings] mp   ON  tb.[object_id]          = mp.[object_id]
JOIN    sys.[pdw_nodes_tables] nt     ON  nt.[name]               = mp.[physical_name]
JOIN    sys.[dm_pdw_nodes_db_column_store_row_group_physical_stats] rg      ON  rg.[object_id]     = nt.
[object_id]
                                                                            AND rg.[pdw_node_id]   = nt.
[pdw_node_id]
                                       AND rg.[distribution_id]    = nt.[distribution_id]
)
select *
from cte;
```

The trim_reason_desc tells whether the rowgroup was trimmed(trim_reason_desc = NO_TRIM implies there was no trimming and row group is of optimal quality). The following trim reasons indicate premature trimming of the rowgroup:

- BULKLOAD: This trim reason is used when the incoming batch of rows for the load had less than 1 million rows. The engine will create compressed row groups if there are greater than 100,000 rows being inserted (as opposed to inserting into the delta store) but sets the trim reason to BULKLOAD. In this scenario, consider increasing your batch load window to accumulate more rows. Also, reevaluate your partitioning scheme to ensure it is not too granular as row groups cannot span partition boundaries.
- MEMORY_LIMITATION: To create row groups with 1 million rows, a certain amount of working memory is required by the engine. When available memory of the loading session is less than the required working memory, row groups get prematurely trimmed. The following sections explain how to estimate memory required and allocate more memory.
- DICTIONARY_SIZE: This trim reason indicates that rowgroup trimming occurred because there was at least one string column with wide and/or high cardinality strings. The dictionary size is limited to 16 MB in memory and once this limit is reached the row group is compressed. If you do run into this situation, consider isolating the problematic column into a separate table.

How to estimate memory requirements

The maximum required memory to compress one rowgroup is approximately

- 72 MB +
- #rows * #columns * 8 bytes +
- #rows * #short-string-columns * 32 bytes +
- #long-string-columns * 16 MB for compression dictionary

where short-string-columns use string data types of <= 32 bytes and long-string-columns use string data types of > 32 bytes.

Long strings are compressed with a compression method designed for compressing text. This compression method uses a *dictionary* to store text patterns. The maximum size of a dictionary is 16 MB. There is only one

For an in-depth discussion of columnstore memory requirements, see the video Azure SQL Data Warehouse scaling: configuration and guidance.

Ways to reduce memory requirements

Use the following techniques to reduce the memory requirements for compressing rowgroups into columnstore indexes.

Use fewer columns

If possible, design the table with fewer columns. When a rowgroup is compressed into the columnstore, the columnstore index compresses each column segment separately. Therefore the memory requirements to compress a rowgroup increase as the number of columns increases.

Use fewer string columns

Columns of string data types require more memory than numeric and date data types. To reduce memory requirements, consider removing string columns from fact tables and putting them in smaller dimension tables.

Additional memory requirements for string compression:

- String data types up to 32 characters can require 32 additional bytes per value.
- String data types with more than 32 characters are compressed using dictionary methods. Each column in the rowgroup can require up to an additional 16 MB to build the dictionary.

Avoid over-partitioning

Columnstore indexes create one or more rowgroups per partition. In SQL Data Warehouse, the number of partitions grows quickly because the data is distributed and each distribution is partitioned. If the table has too many partitions, there might not be enough rows to fill the rowgroups. The lack of rows does not create memory pressure during compression, but it leads to rowgroups that do not achieve the best columnstore query performance.

Another reason to avoid over-partitioning is there is a memory overhead for loading rows into a columnstore index on a partitioned table. During a load, many partitions could receive the incoming rows, which are held in memory until each partition has enough rows to be compressed. Having too many partitions creates additional memory pressure.

Simplify the load query

The database shares the memory grant for a query among all the operators in the query. When a load query has complex sorts and joins, the memory available for compression is reduced.

Design the load query to focus only on loading the query. If you need to run transformations on the data, run them separate from the load query. For example, stage the data in a heap table, run the transformations, and then load the staging table into the columnstore index. You can also load the data first and then use the MPP system to transform the data.

Adjust MAXDOP

Each distribution compresses rowgroups into the columnstore in parallel when there is more than one CPU core available per distribution. The parallelism requires additional memory resources, which can lead to memory pressure and rowgroup trimming.

To reduce memory pressure, you can use the MAXDOP query hint to force the load operation to run in serial mode within each distribution.

```
CREATE TABLE MyFactSalesQuota
WITH (DISTRIBUTION = ROUND_ROBIN)
AS SELECT * FROM FactSalesQUota
OPTION (MAXDOP 1);
```

Ways to allocate more memory

DWU size and the user resource class together determine how much memory is available for a user query. To increase the memory grant for a load query, you can either increase the number of DWUs or increase the resource class.

- To increase the DWUs, see How do I scale performance?
- To change the resource class for a query, see Change a user resource class example.

For example, on DWU 100 a user in the smallrc resource class can use 100 MB of memory for each distribution. For the details, see Concurrency in SQL Data Warehouse.

Suppose you determine that you need 700 MB of memory to get high-quality rowgroup sizes. These examples show how you can run the load query with enough memory.

- Using DWU 1000 and mediumrc, your memory grant is 800 MB
- Using DWU 600 and largerc, your memory grant is 800 MB.

Next steps

To find more ways to improve performance in SQL Data Warehouse, see the Performance overview.

Monitor your workload using DMVs

12/15/2017 • 7 min to read • Edit Online

This article describes how to use Dynamic Management Views (DMVs) to monitor your workload and investigate query execution in Azure SQL Data Warehouse.

Permissions

To query the DMVs in this article, you need either VIEW DATABASE STATE or CONTROL permission. Usually granting VIEW DATABASE STATE is the preferred permission as it is much more restrictive.

```
GRANT VIEW DATABASE STATE TO myuser;
```

Monitor connections

All logins to SQL Data Warehouse are logged to sys.dm_pdw_exec_sessions. This DMV contains the last 10,000 logins. The session_id is the primary key and is assigned sequentially for each new logon.

```
-- Other Active Connections
SELECT * FROM sys.dm_pdw_exec_sessions where status <> 'Closed' and session_id <> session_id();
```

Monitor query execution

All queries executed on SQL Data Warehouse are logged to sys.dm_pdw_exec_requests. This DMV contains the last 10,000 queries executed. The request_id uniquely identifies each query and is the primary key for this DMV. The request_id is assigned sequentially for each new query and is prefixed with QID, which stands for query ID. Querying this DMV for a given session_id shows all queries for a given logon.

> **NOTE**
> Stored procedures use multiple Request IDs. Request IDs are assigned in sequential order.

Here are steps to follow to investigate query execution plans and times for a particular query.

STEP 1: Identify the query you wish to investigate

```
-- Monitor active queries
SELECT *
FROM sys.dm_pdw_exec_requests
WHERE status not in ('Completed','Failed','Cancelled')
  AND session_id <> session_id()
ORDER BY submit_time DESC;

-- Find top 10 queries longest running queries
SELECT TOP 10 *
FROM sys.dm_pdw_exec_requests
ORDER BY total_elapsed_time DESC;

-- Find a query with the Label 'My Query'
-- Use brackets when querying the label column, as it it a key word
SELECT  *
FROM    sys.dm_pdw_exec_requests
WHERE   [label] = 'My Query';
```

From the preceding query results, **note the Request ID** of the query that you would like to investigate.

Queries in the **Suspended** state are being queued due to concurrency limits. These queries also appear in the sys.dm_pdw_waits waits query with a type of UserConcurrencyResourceType. See Concurrency and workload management for more details on concurrency limits. Queries can also wait for other reasons such as for object locks. If your query is waiting for a resource, see Investigating queries waiting for resources further down in this article.

To simplify the lookup of a query in the sys.dm_pdw_exec_requests table, use LABEL to assign a comment to your query that can be looked up in the sys.dm_pdw_exec_requests view.

```
-- Query with Label
SELECT *
FROM sys.tables
OPTION (LABEL = 'My Query')
;
```

STEP 2: Investigate the query plan

Use the Request ID to retrieve the query's distributed SQL (DSQL) plan from sys.dm_pdw_request_steps.

```
-- Find the distributed query plan steps for a specific query.
-- Replace request_id with value from Step 1.

SELECT * FROM sys.dm_pdw_request_steps
WHERE request_id = 'QID####'
ORDER BY step_index;
```

When a DSQL plan is taking longer than expected, the cause can be a complex plan with many DSQL steps or just one step taking a long time. If the plan is many steps with several move operations, consider optimizing your table distributions to reduce data movement. The Table distribution article explains why data must be moved to solve a query and explains some distribution strategies to minimize data movement.

To investigate further details about a single step, the *operation_type* column of the long-running query step and note the **Step Index**:

- Proceed with Step 3a for **SQL operations**: OnOperation, RemoteOperation, ReturnOperation.
- Proceed with Step 3b for **Data Movement operations**: ShuffleMoveOperation, BroadcastMoveOperation, TrimMoveOperation, PartitionMoveOperation, MoveOperation, CopyOperation.

STEP 3a: Investigate SQL on the distributed databases

information of the query step on all of the distributed databases.

```
-- Find the distribution run times for a SQL step.
-- Replace request_id and step_index with values from Step 1 and 3.

SELECT * FROM sys.dm_pdw_sql_requests
WHERE request_id = 'QID####' AND step_index = 2;
```

When the query step is running, DBCC PDW_SHOWEXECUTIONPLAN can be used to retrieve the SQL Server estimated plan from the SQL Server plan cache for the step running on a particular distribution.

```
-- Find the SQL Server execution plan for a query running on a specific SQL Data Warehouse Compute or Control
node.
-- Replace distribution_id and spid with values from previous query.

DBCC PDW_SHOWEXECUTIONPLAN(1, 78);
```

STEP 3b: Investigate data movement on the distributed databases

Use the Request ID and the Step Index to retrieve information about a data movement step running on each distribution from sys.dm_pdw_dms_workers.

```
-- Find the information about all the workers completing a Data Movement Step.
-- Replace request_id and step_index with values from Step 1 and 3.

SELECT * FROM sys.dm_pdw_dms_workers
WHERE request_id = 'QID####' AND step_index = 2;
```

- Check the *total_elapsed_time* column to see if a particular distribution is taking significantly longer than others for data movement.
- For the long-running distribution, check the *rows_processed* column to see if the number of rows being moved from that distribution is significantly larger than others. If so, this may indicate skew of your underlying data.

If the query is running, DBCC PDW_SHOWEXECUTIONPLAN can be used to retrieve the SQL Server estimated plan from the SQL Server plan cache for the currently running SQL Step within a particular distribution.

```
-- Find the SQL Server estimated plan for a query running on a specific SQL Data Warehouse Compute or Control
node.
-- Replace distribution_id and spid with values from previous query.

DBCC PDW_SHOWEXECUTIONPLAN(55, 238);
```

Monitor waiting queries

If you discover that your query is not making progress because it is waiting for a resource, here is a query that shows all the resources a query is waiting for.

```
-- Find queries
-- Replace request_id with value from Step 1.

SELECT waits.session_id,
      waits.request_id,
      requests.command,
      requests.status,
      requests.start_time,
      waits.type,
      waits.state,
      waits.object_type,
      waits.object_name
FROM   sys.dm_pdw_waits waits
   JOIN  sys.dm_pdw_exec_requests requests
   ON waits.request_id=requests.request_id
WHERE waits.request_id = 'QID####'
ORDER BY waits.object_name, waits.object_type, waits.state;
```

If the query is actively waiting on resources from another query, then the state will be **AcquireResources**. If the query has all the required resources, then the state will be **Granted**.

Monitor tempdb

High tempdb utilization can be the root cause for slow performance and out of memory issues. Consider scaling your data warehouse if you find tempdb reaching its limits during query execution. The following describes how to identify tempdb usage per query on each node.

Create the following view to associate the appropriate node id for sys.dm_pdw_sql_requests. This will enable you to leverage other pass-through DMVs and join those tables with sys.dm_pdw_sql_requests.

```
-- sys.dm_pdw_sql_requests with the correct node id
CREATE VIEW sql_requests AS
(SELECT
       sr.request_id,
       sr.step_index,
       (CASE
              WHEN (sr.distribution_id = -1 ) THEN
              (SELECT pdw_node_id FROM sys.dm_pdw_nodes WHERE type = 'CONTROL')
              ELSE d.pdw_node_id END) AS pdw_node_id,
       sr.distribution_id,
       sr.status,
       sr.error_id,
       sr.start_time,
       sr.end_time,
       sr.total_elapsed_time,
       sr.row_count,
       sr.spid,
       sr.command
FROM sys.pdw_distributions AS d
RIGHT JOIN sys.dm_pdw_sql_requests AS sr ON d.distribution_id = sr.distribution_id)
```

Run the following query to monitor tempdb:

```
-- Monitor tempdb
SELECT
    sr.request_id,
    ssu.session_id,
    ssu.pdw_node_id,
    sr.command,
    sr.total_elapsed_time,
    es.login_name AS 'LoginName',
    DB_NAME(ssu.database_id) AS 'DatabaseName',
    (es.memory_usage * 8) AS 'MemoryUsage (in KB)',
    (ssu.user_objects_alloc_page_count * 8) AS 'Space Allocated For User Objects (in KB)',
    (ssu.user_objects_dealloc_page_count * 8) AS 'Space Deallocated For User Objects (in KB)',
    (ssu.internal_objects_alloc_page_count * 8) AS 'Space Allocated For Internal Objects (in KB)',
    (ssu.internal_objects_dealloc_page_count * 8) AS 'Space Deallocated For Internal Objects (in KB)',
    CASE es.is_user_process
    WHEN 1 THEN 'User Session'
    WHEN 0 THEN 'System Session'
    END AS 'SessionType',
    es.row_count AS 'RowCount'
FROM sys.dm_pdw_nodes_db_session_space_usage AS ssu
    INNER JOIN sys.dm_pdw_nodes_exec_sessions AS es ON ssu.session_id = es.session_id AND ssu.pdw_node_id =
es.pdw_node_id
    INNER JOIN sys.dm_pdw_nodes_exec_connections AS er ON ssu.session_id = er.session_id AND ssu.pdw_node_id =
er.pdw_node_id
    INNER JOIN sql_requests AS sr ON ssu.session_id = sr.spid AND ssu.pdw_node_id = sr.pdw_node_id
WHERE DB_NAME(ssu.database_id) = 'tempdb'
    AND es.session_id <> @@SPID
    AND es.login_name <> 'sa'
ORDER BY sr.request_id;
```

Monitor memory

Memory can be the root cause for slow performance and out of memory issues. Consider scaling your data warehouse if you find SQL Server memory usage reaching its limits during query execution.

The following query returns SQL Server memory usage and memory pressure per node:

```
-- Memory consumption
SELECT
  pc1.cntr_value as Curr_Mem_KB,
  pc1.cntr_value/1024.0 as Curr_Mem_MB,
  (pc1.cntr_value/1048576.0) as Curr_Mem_GB,
  pc2.cntr_value as Max_Mem_KB,
  pc2.cntr_value/1024.0 as Max_Mem_MB,
  (pc2.cntr_value/1048576.0) as Max_Mem_GB,
  pc1.cntr_value * 100.0/pc2.cntr_value AS Memory_Utilization_Percentage,
  pc1.pdw_node_id
FROM
-- pc1: current memory
sys.dm_pdw_nodes_os_performance_counters AS pc1
-- pc2: total memory allowed for this SQL instance
JOIN sys.dm_pdw_nodes_os_performance_counters AS pc2
ON pc1.object_name = pc2.object_name AND pc1.pdw_node_id = pc2.pdw_node_id
WHERE
pc1.counter_name = 'Total Server Memory (KB)'
AND pc2.counter_name = 'Target Server Memory (KB)'
```

Monitor transaction log size

The following query returns the transaction log size on each distribution. If one of the log files is reaching 160GB, you should consider scaling up your instance or limiting your transaction size.

```
-- Transaction log size
SELECT
  instance_name as distribution_db,
  cntr_value*1.0/1048576 as log_file_size_used_GB,
  pdw_node_id
FROM sys.dm_pdw_nodes_os_performance_counters
WHERE
instance_name like 'Distribution_%'
AND counter_name = 'Log File(s) Used Size (KB)'
AND counter_name = 'Target Server Memory (KB)'
```

Monitor transaction log rollback

If your queries are failing or taking a long time to proceed, you can check and monitor if you have any transactions rolling back.

```
-- Monitor rollback
SELECT
    SUM(CASE WHEN t.database_transaction_next_undo_lsn IS NOT NULL THEN 1 ELSE 0 END),
    t.pdw_node_id,
    nod.[type]
FROM sys.dm_pdw_nodes_tran_database_transactions t
JOIN sys.dm_pdw_nodes nod ON t.pdw_node_id = nod.pdw_node_id
GROUP BY t.pdw_node_id, nod.[type]
```

Next steps

See System views for more information on DMVs. See SQL Data Warehouse best practices for more information about best practices

Troubleshooting Azure SQL Data Warehouse

12/18/2017 • 5 min to read • Edit Online

This topic lists some of the more common troubleshooting questions we hear from our customers.

Connecting

ISSUE	RESOLUTION
Login failed for user 'NT AUTHORITY\ANONYMOUS LOGON'. (Microsoft SQL Server, Error: 18456)	This error occurs when an AAD user tries to connect to the master database, but does not have a user in master. To correct this issue either specify the SQL Data Warehouse you wish to connect to at connection time or add the user to the master database. See Security overview article for more details.
The server principal "MyUserName" is not able to access the database "master" under the current security context. Cannot open user default database. Login failed. Login failed for user 'MyUserName'. (Microsoft SQL Server, Error: 916)	This error occurs when an AAD user tries to connect to the master database, but does not have a user in master. To correct this issue either specify the SQL Data Warehouse you wish to connect to at connection time or add the user to the master database. See Security overview article for more details.
CTAIP error	This error can occur when a login has been created on the SQL server master database, but not in the SQL Data Warehouse database. If you encounter this error, take a look at the Security overview article. This article explains how to create create a login and user on master and then how to create a user in the SQL Data Warehouse database.
Blocked by Firewall	Azure SQL databases are protected by server and database level firewalls to ensure only known IP addresses have access to a database. The firewalls are secure by default, which means that you must explicitly enable and IP address or range of addresses before you can connect. To configure your firewall for access, follow the steps in Configure server firewall access for your client IP in the Provisioning instructions.
Cannot connect with tool or driver	SQL Data Warehouse recommends using SSMS, SSDT for Visual Studio, or sqlcmd to query your data. For more details on drivers and connecting to SQL Data Warehouse, see Drivers for Azure SQL Data Warehouse and Connect to Azure SQL Data Warehouse articles.

Tools

ISSUE	RESOLUTION
Visual Studio object explorer is missing AAD users	This is a known issue. As a workaround, view the users in sys.database_principals. See Authentication to Azure SQL Data Warehouse to learn more about using Azure Active Directory with SQL Data Warehouse.

ISSUE	RESOLUTION
Manual scripting, using the scripting wizard, or connecting via SSMS is slow, hung, or producing errors	Please make sure that users have been created in the master database. In scripting options, also make sure that the engine edition is set as "Microsoft Azure SQL Data Warehouse Edition" and engine type is "Microsoft Azure SQL Database".

Performance

ISSUE	RESOLUTION
Query performance troubleshooting	If you are trying to troubleshoot a particular query, start with Learning how to monitor your queries.
Poor query performance and plans often is a result of missing statistics	The most common cause of poor performance is lack of statistics on your tables. See Maintaining table statistics for details on how to create statistics and why they are critical to your performance.
Low concurrency / queries queued	Understanding Workload management is important in order to understand how to balance memory allocation with concurrency.
How to implement best practices	The best place to start to learn ways to improve query performance is SQL Data Warehouse best practices article.
How to improve performance with scaling	Sometimes the solution to improving performance is to simply add more compute power to your queries by Scaling your SQL Data Warehouse.
Poor query performance as a result of poor index quality	Some times queries can slowdown because of Poor columnstore index quality. See this article for more information and how to Rebuild indexes to improve segment quality.

System management

ISSUE	RESOLUTION
Msg 40847: Could not perform the operation because server would exceed the allowed Database Transaction Unit quota of 45000.	Either reduce the DWU of the database you are trying to create or request a quota increase.
Investigating space utilization	See Table sizes to understand the space utilization of your system.
Help with managing tables	See the Table overview article for help with managing your tables. This article also includes links into more detailed topics like Table data types, Distributing a table, Indexing a table, Partitioning a table, Maintaining table statistics and Temporary tables.
Transparent data encryption (TDE) progress bar is not updating in the Azure Portal	You can view the state of TDE via powershell.

PolyBase

ISSUE	RESOLUTION
Load fails because of large rows	Currently large row support is not available for Polybase. This means that if your table contains VARCHAR(MAX), NVARCHAR(MAX) or VARBINARY(MAX), External tables cannot be used to load your data. Loads for large rows is currently only supported through Azure Data Factory (with BCP), Azure Stream Analytics, SSIS, BCP or the .NET SQLBulkCopy class. PolyBase support for large rows will be added in a future release.
bcp load of table with MAX data type is failing	There is a known issue which requires that VARCHAR(MAX), NVARCHAR(MAX) or VARBINARY(MAX) be placed at the end of the table in some scenarios. Try moving your MAX columns to the end of the table.

Differences from SQL Database

ISSUE	RESOLUTION
Unsupported SQL Database features	See Unsupported table features.
Unsupported SQL Database data types	See Unsupported data types.
DELETE and UPDATE limitations	See UPDATE workarounds, DELETE workarounds and Using CTAS to work around unsupported UPDATE and DELETE syntax.
MERGE statement is not supported	See MERGE workarounds.
Stored procedure limitations	See Stored procedure limitations to understand some of the limitations of stored procedures.
UDFs do not support SELECT statements	This is a current limitation of our UDFs. See CREATE FUNCTION for the syntax we support.

Next steps

If you are were unable to find a solution to your issue above, here are some other resources you can try.

- Blogs
- Feature requests
- Videos
- CAT team blogs
- Create support ticket
- MSDN forum
- Stack Overflow forum
- Twitter

Design decisions and coding techniques for SQL Data Warehouse

6/27/2017 • 1 min to read • Edit Online

Take a look through these development articles to better understand key design decisions, recommendations and coding techniques for SQL Data Warehouse.

Key design decisions

The following articles highlight some of the key concepts and design decisions you will need to understand for the development of your distributed data warehouse using SQL Data Warehouse:

- connections
- concurrency
- transactions
- user-defined schemas
- table distribution
- table indexes
- table partitions
- CTAS
- statistics

Development recommendations and coding techniques

These articles highlight specific coding techniques, tips and recommendations for developing your SQL Data Warehouse:

- stored procedures
- labels
- views
- temporary tables
- dynamic SQL
- looping
- group by options
- variable assignment

Next steps

Once you have been through the development articles take a look through the Transact-SQL reference page for more details on the supported syntax for SQL Data Warehouse.

Data warehouse workload

6/27/2017 • 4 min to read • Edit Online

A data warehouse workload refers to all of the operations that transpire against a data warehouse. The data warehouse workload encompasses the entire process of loading data into the warehouse, performing analysis and reporting on the data warehouse, managing data in the data warehouse, and exporting data from the data warehouse. The depth and breadth of these components are often commensurate with the maturity level of the data warehouse.

New to data warehousing?

A data warehouse is a collection of data that is loaded from one or more data sources and is used to perform business intelligence tasks such as reporting and data analysis.

Data warehouses are characterized by queries that scan larger numbers of rows, large ranges of data and may return relatively large results for the purposes of analysis and reporting. Data warehouses are also characterized by relatively large data loads versus small transaction-level inserts/updates/deletes.

- A data warehouse performs best when the data is stored in a way that optimizes queries that need to scan large numbers of rows or large ranges of data. This type of scanning works best when the data is stored and searched by columns, instead of by rows.

> **NOTE**
> The in-memory columnstore index, which uses column storage, provides up to 10x compression gains and 100x query performance gains over traditional binary trees for reporting and analytics queries. We consider columnstore indexes as the standard for storing and scanning large data in a data warehouse.

- A data warehouse has different requirements than a system that optimizes for online transaction processing (OLTP). The OLTP system has many insert, update, and delete operations. These operations seek to specific rows in the table. Table seeks perform best when the data is stored in a row-by-row manner. The data can be sorted and quickly searched with a divide and conquer approach called a binary tree or btree search.

Data loading

Data loading is a big part of the data warehouse workload. Businesses usually have a busy OLTP system that tracks changes throughout the day when customers are generating business transactions. Periodically, often at night during a maintenance window, the transactions are moved or copied to the data warehouse. Once the data is in the data warehouse, analysts can perform analysis and make business decisions on the data.

- Traditionally, the process of loading is called ETL for Extract, Transform, and Load. Data usually needs to be transformed so it is consistent with other data in the data warehouse. Previously, businesses used dedicated ETL servers to perform the transformations. Now, with such fast massively parallel processing you can load data into SQL Data Warehouse first, and then perform the transformations. This process is called Extract, Load, and Transform (ELT), and is becoming a new standard for the data warehouse workload.

NOTE

With SQL Server 2016, you can now perform analytics in real-time on an OLTP table. This does not replace the need for a data warehouse to store and analyze data, but it does provide a way to perform analysis in real-time.

Reporting and analysis queries

Reporting and analysis queries are often categorized into small, medium and large based on a number of criteria, but usually it is based on time. In most data warehouses, there is a mixed workload of fast-running versus long-running queries. In each case, it is important to determine this mix and to determine its frequency (hourly, daily, month-end, quarter-end, and so on). It is important to understand that the mixed query workload, coupled with concurrency, lead to proper capacity planning for a data warehouse.

- Capacity planning can be a complex task for a mixed query workload, especially when you need a long lead time to add capacity to the data warehouse. SQL Data Warehouse removes the urgency of capacity planning since you can grow and shrink compute capacity at any time, and since storage and compute capacity are independently sized.

Data management

Managing data is important, especially when you know you might run out of disk space in the near future. Data warehouses usually divide the data into meaningful ranges, which are stored as partitions in a table. All SQL Server-based products let you move partitions in and out of the table. This partition switching lets you move older data to less expensive storage and keep the latest data available on online storage.

- Columnstore indexes support partitioned tables. For columnstore indexes, partitioned tables are used for data management and archival. For tables stored row-by-row, partitions have a larger role in query performance.
- PolyBase plays an important role in managing data. Using PolyBase, you have the option to archive older data to Hadoop or Azure blob storage. This provides lots of options since the data is still online. It might take longer to retrieve data from Hadoop, but the tradeoff of retrieval time might outweigh the storage cost.

Exporting data

One way to make data available for reports and analysis is to send data from the data warehouse to servers dedicated for running reports and analysis. These servers are called data marts. For example, you could pre-process report data and then export it from the data warehouse to many servers around the world, to make it broadly available for customers and analysts.

- For generating reports, each night you could populate read-only reporting servers with a snapshot of the daily data. This gives higher bandwidth for customers while lowering the compute resource needs on the data warehouse. From a security aspect, data marts allow you to reduce the number of users who have access to the data warehouse.
- For analytics, you can either build an analysis cube on the data warehouse and run analysis against the data warehouse, or pre-process data and export it to the analysis server for further analytics.

Next steps

Now that you know a bit about SQL Data Warehouse, learn how to quickly create a SQL Data Warehouse and load sample data.

Introduction to designing tables in Azure SQL Data Warehouse

1/8/2018 • 10 min to read • Edit Online

Learn key concepts for designing tables in Azure SQL Data Warehouse.

Determining table category

A star schema organizes data into fact and dimension tables. Some tables are used for integration or staging data before it moves to a fact or dimension table. As you design a table, decide whether the table data belongs in a fact, dimension, or integration table. This decision informs the appropriate table structure and distribution.

- **Fact tables** contain quantitative data that are commonly generated in a transactional system and then loaded into the data warehouse. For example, a retail business generates sales transactions every day and then loads the data to a data warehouse fact table for analysis.
- **Dimension tables** contain attribute data that might change but usually changes infrequently. For example, a customer's name and address is stored in a dimension table and updated only when the customer's profile changes. To minimize the size of a large fact table, the customer's name and address do not need to be in every row of a fact table. Instead, the fact table and the dimension table can share a customer ID. A query can join the two tables to associate a customer's profile and transactions.
- **Integration tables** provide a place for integrating or staging data. These tables can be creating as regular tables, external tables, or temporary tables. For example, you can load data to a staging table, perform transformations on the data in staging, and then insert the data into a production table.

Schema and table names

In SQL Data Warehouse, a data warehouse is a type of database. All of the tables in the data warehouse are contained within the same database. You cannot join tables across multiple data warehouses. This behavior is different from SQL Server which supports cross-database joins.

In a SQL Server database, you might use fact, dim, or integrate for the schema names. If you are transferring a SQL Server database to SQL Data Warehouse, it works best to migrate store all of the fact, dimension, and integration tables to one schema in SQL Data Warehouse. For example, you could store all the tables in the WideWorldImportersDW sample data warehouse within one schema called WWI. The following code creates a user-defined schema called WWI.

```
CREATE SCHEMA WWI;
```

To show the organization of the tables in SQL Data Warehouse, use fact, dim, and int as prefixes to the table names. The following table shows some of the schema and table names for WideWorldImportersDW. It compares the names in SQL Server and SQL Data Warehouse.

WWI DIMENSION TABLES	SQL SERVER	SQL DATA WAREHOUSE
City	Dimension.City	WWI.DimCity
Customer	Dimension.Customer	WWI.DimCustomer

WWI DIMENSION TABLES	SQL SERVER	SQL DATA WAREHOUSE
Date	Dimension.Date	WWI.DimDate

WWI FACT TABLES	SQL SERVER	SQL DATA WAREHOUSE
Order	Fact.Order	WWI.FactOrder
Sale	Fact.Sale	WWI.FactSale
Purchase	Fact	WWI.FactPurchase

Table definition

The following concepts explain key aspects of defining tables.

Standard table

A standard table is stored in Azure Storage as part of the data warehouse. The table and the data persist regardless of whether a session is open. This example creates a table with two columns.

```
CREATE TABLE MyTable (col1 int, col2 int );
```

Temporary table

A temporary table only exists for the duration of the session. You can use a temporary table to prevent other uses from seeing temporary results and also to reduce the need for cleanup. Since temporary tables also utilize local storage, they can offer faster performance for some operations. For more information, see Temporary tables.

External table

An external table points to data located in Azure blob storage or Azure Data Lake Store. When used in conjunction with the CREATE TABLE AS SELECT statement, selecting from an external table imports data into SQL Data Warehouse. External tables are therefore useful for loading data. For a loading tutorial, see Use PolyBase to load data from Azure blob storage.

Data types

SQL Data Warehouse supports the most commonly used data types. For a list of the supported data types, see data types in the CREATE TABLE statement. Minimizing the size of data types helps to improve query performance. For guidance on data types, see Data types.

Distributed tables

A fundamental feature of SQL Data Warehouse is the way it can store tables across 60 distributed locations, called distributions, in the distributed system. SQL Data Warehouse can store a table in one of these three ways:

- **Round-robin** stores table rows randomly, but evenly across the distributions. The round-robin table achieves fast loading performance, but requires more data movement than the other methods for queries that join columns.
- **Hash** distribution distributes rows based on the value in the distribution column. The hash-distributed table has the most potential to achieve high performance for query joins on large tables. There are several factors that affect the choice of the distribution column. For more information, see Distributed tables.
- **Replicated** tables make a full copy of the table available on every Compute node. Queries run fast on replicated tables since joins on replicated tables do not require data movement. Replication requires extra

The table category often determines which option to choose for distributing the table.

TABLE CATEGORY	RECOMMENDED DISTRIBUTION OPTION
Fact	Use hash-distribution with clustered columnstore index. Performance improves when two hash tables are joined on the same distribution column.
Dimension	Use replicated for smaller tables. If tables are too large to store on each Compute node, use hash-distributed.
Staging	Use round-robin for the staging table. The load with CTAS is fast. Once the data is in the staging table, use INSERT...SELECT to move the data to a production tables.

Table partitions

A partitioned table stores and performs operations on the table rows according to data ranges. For example, a table could be partitioned by day, month, or year. You can improve query performance partition elimination, which limits a query scan to data within a partition. You can also maintain the data through partition switching. Since the data in SQL Data Warehouse is already distributed, too many partitions can slow query performance. For more information, see Partitioning guidance.

Columnstore indexes

By default, SQL Data Warehouse stores a table as a clustered columnstore index. This form of data storage achieves high data compression and query performance on large tables. The clustered columnstore index is usually the best choice, but in some cases a clustered index or a heap is the appropriate storage structure.

For a list of columnstore features, see What's new for columnstore indexes. To improve columnstore index performance, see Maximizing rowgroup quality for columnstore indexes.

Statistics

The query optimizer uses column-level statistics when it creates the plan for executing a query. To improve query performance, it's important to create statistics on individual columns, especially columns used in query joins. Creating and updating statistics does not happen automatically. Create statistics after creating a table. Update statistics after a significant number of rows are added or changed. For example, update statistics after a load. For more information, see Statistics guidance.

Ways to create tables

You can create a table as a new empty table. You can also create and populate a table with the results of a select statement. The following are the T-SQL commands for creating a table.

T-SQL STATEMENT	DESCRIPTION
CREATE TABLE	Creates an empty table by defining all the table columns and options.
CREATE EXTERNAL TABLE	Creates an external table. The definition of the table is stored in SQL Data Warehouse. The table data is stored in Azure Blob storage or Azure Data Lake Store.

T-SQL STATEMENT	DESCRIPTION
CREATE TABLE AS SELECT	Populates a new table with the results of a select statement. The table columns and data types are based on the select statement results. To import data, this statement can select from an external table.
CREATE EXTERNAL TABLE AS SELECT	Creates a new external table by exporting the results of a select statement to an external location. The location is either Azure Blob storage or Azure Data Lake Store.

Aligning source data with the data warehouse

Data warehouse tables are populated by loading data from another data source. To perform a successful load, the number and data types of the columns in the source data must align with the table definition in the data warehouse. Getting the data to align might be the hardest part of designing your tables.

If data is coming from multiple data stores, you can bring the data into the data warehouse and store it in an integration table. Once data is in the integrtion table, you can use the power of SQL Data Warehouse to perform transformation operations.

Unsupported table features

SQL Data Warehouse supports many, but not all, of the table features offered by other databases. The following list shows some of the table features that are not supported in SQL Data Warehouse.

- Primary key, Foreign keys, Unique, Check Table Constraints
- Computed Columns
- Indexed Views
- Sequence
- Sparse Columns
- Implement with Identity.
- Synonyms
- Triggers
- Unique Indexes
- User-Defined Types

Table size queries

One simple way to identify space and rows consumed by a table in each of the 60 distributions, is to use [DBCC PDW_SHOWSPACEUSED][DBCC PDW_SHOWSPACEUSED].

```
DBCC PDW_SHOWSPACEUSED('dbo.FactInternetSales');
```

However, using DBCC commands can be quite limiting. Dynamic management views (DMVs) show more detail than DBCC commands. Start by creating this view.

```
CREATE VIEW dbo.vTableSizes
AS
WITH base
AS
(
```

```
, DB_NAME()                                                                 AS  [database_name]
, s.name                                                                    AS  [schema_name]
, t.name                                                                    AS  [table_name]
, QUOTENAME(s.name)+'.'+QUOTENAME(t.name)                                   AS  [two_part_name]
, nt.[name]                                                                 AS  [node_table_name]
, ROW_NUMBER() OVER(PARTITION BY nt.[name] ORDER BY (SELECT NULL))          AS  [node_table_name_seq]
, tp.[distribution_policy_desc]                                             AS  [distribution_policy_name]
, c.[name]                                                                  AS  [distribution_column]
, nt.[distribution_id]                                                      AS  [distribution_id]
, i.[type]                                                                  AS  [index_type]
, i.[type_desc]                                                             AS  [index_type_desc]
, nt.[pdw_node_id]                                                          AS  [pdw_node_id]
, pn.[type]                                                                 AS  [pdw_node_type]
, pn.[name]                                                                 AS  [pdw_node_name]
, di.name                                                                   AS  [dist_name]
, di.position                                                               AS  [dist_position]
, nps.[partition_number]                                                    AS  [partition_nmbr]
, nps.[reserved_page_count]                                                 AS  [reserved_space_page_count]
, nps.[reserved_page_count] - nps.[used_page_count]                         AS  [unused_space_page_count]
, nps.[in_row_data_page_count]
    + nps.[row_overflow_used_page_count]
    + nps.[lob_used_page_count]                                             AS  [data_space_page_count]
, nps.[reserved_page_count]
 - (nps.[reserved_page_count] - nps.[used_page_count])
 - ([in_row_data_page_count]
        + [row_overflow_used_page_count]+[lob_used_page_count])             AS  [index_space_page_count]
, nps.[row_count]                                                           AS  [row_count]
from
    sys.schemas s
INNER JOIN sys.tables t
    ON s.[schema_id] = t.[schema_id]
INNER JOIN sys.indexes i
    ON  t.[object_id] = i.[object_id]
    AND i.[index_id] <= 1
INNER JOIN sys.pdw_table_distribution_properties tp
    ON t.[object_id] = tp.[object_id]
INNER JOIN sys.pdw_table_mappings tm
    ON t.[object_id] = tm.[object_id]
INNER JOIN sys.pdw_nodes_tables nt
    ON tm.[physical_name] = nt.[name]
INNER JOIN sys.dm_pdw_nodes pn
    ON  nt.[pdw_node_id] = pn.[pdw_node_id]
INNER JOIN sys.pdw_distributions di
    ON  nt.[distribution_id] = di.[distribution_id]
INNER JOIN sys.dm_pdw_nodes_db_partition_stats nps
    ON nt.[object_id] = nps.[object_id]
    AND nt.[pdw_node_id] = nps.[pdw_node_id]
    AND nt.[distribution_id] = nps.[distribution_id]
LEFT OUTER JOIN (select * from sys.pdw_column_distribution_properties where distribution_ordinal = 1) cdp
    ON t.[object_id] = cdp.[object_id]
LEFT OUTER JOIN sys.columns c
    ON cdp.[object_id] = c.[object_id]
    AND cdp.[column_id] = c.[column_id]
)
, size
AS
(
SELECT
   [execution_time]
,  [database_name]
,  [schema_name]
,  [table_name]
,  [two_part_name]
,  [node_table_name]
,  [node_table_name_seq]
,  [distribution_policy_name]
,  [distribution_column]
```

```
,  [index_type_desc]
,  [pdw_node_id]
,  [pdw_node_type]
,  [pdw_node_name]
,  [dist_name]
,  [dist_position]
,  [partition_nmbr]
,  [reserved_space_page_count]
,  [unused_space_page_count]
,  [data_space_page_count]
,  [index_space_page_count]
,  [row_count]
,  ([reserved_space_page_count] * 8.0)                              AS [reserved_space_KB]
,  ([reserved_space_page_count] * 8.0)/1000                         AS [reserved_space_MB]
,  ([reserved_space_page_count] * 8.0)/1000000                      AS [reserved_space_GB]
,  ([reserved_space_page_count] * 8.0)/1000000000                   AS [reserved_space_TB]
,  ([unused_space_page_count]   * 8.0)                              AS [unused_space_KB]
,  ([unused_space_page_count]   * 8.0)/1000                         AS [unused_space_MB]
,  ([unused_space_page_count]   * 8.0)/1000000                      AS [unused_space_GB]
,  ([unused_space_page_count]   * 8.0)/1000000000                   AS [unused_space_TB]
,  ([data_space_page_count]     * 8.0)                              AS [data_space_KB]
,  ([data_space_page_count]     * 8.0)/1000                         AS [data_space_MB]
,  ([data_space_page_count]     * 8.0)/1000000                      AS [data_space_GB]
,  ([data_space_page_count]     * 8.0)/1000000000                   AS [data_space_TB]
,  ([index_space_page_count]  * 8.0)                                AS [index_space_KB]
,  ([index_space_page_count]  * 8.0)/1000                           AS [index_space_MB]
,  ([index_space_page_count]  * 8.0)/1000000                        AS [index_space_GB]
,  ([index_space_page_count]  * 8.0)/1000000000                     AS [index_space_TB]
FROM base
)
SELECT *
FROM size
;
```

Table space summary

This query returns the rows and space by table. It allows you to see which tables are your largest tables and whether they are round-robin, replicated, or hash -distributed. For hash-distributed tables, the query shows the distribution column. In most cases your largest tables should be hash-distributed with a clustered columnstore index.

```
SELECT
     database_name
,    schema_name
,    table_name
,    distribution_policy_name
,      distribution_column
,    index_type_desc
,    COUNT(distinct partition_nmbr) as nbr_partitions
,    SUM(row_count)                 as table_row_count
,    SUM(reserved_space_GB)         as table_reserved_space_GB
,    SUM(data_space_GB)             as table_data_space_GB
,    SUM(index_space_GB)            as table_index_space_GB
,    SUM(unused_space_GB)           as table_unused_space_GB
FROM
    dbo.vTableSizes
GROUP BY
     database_name
,    schema_name
,    table_name
,    distribution_policy_name
,      distribution_column
,    index_type_desc
ORDER BY
    table_reserved_space_GB desc
;
```

Table space by distribution type

```
SELECT
     distribution_policy_name
,    SUM(row_count)                as table_type_row_count
,    SUM(reserved_space_GB)        as table_type_reserved_space_GB
,    SUM(data_space_GB)            as table_type_data_space_GB
,    SUM(index_space_GB)           as table_type_index_space_GB
,    SUM(unused_space_GB)          as table_type_unused_space_GB
FROM dbo.vTableSizes
GROUP BY distribution_policy_name
;
```

Table space by index type

```
SELECT
     index_type_desc
,    SUM(row_count)                as table_type_row_count
,    SUM(reserved_space_GB)        as table_type_reserved_space_GB
,    SUM(data_space_GB)            as table_type_data_space_GB
,    SUM(index_space_GB)           as table_type_index_space_GB
,    SUM(unused_space_GB)          as table_type_unused_space_GB
FROM dbo.vTableSizes
GROUP BY index_type_desc
;
```

Distribution space summary

```
SELECT
    distribution_id
,    SUM(row_count)                as total_node_distribution_row_count
,    SUM(reserved_space_MB)        as total_node_distribution_reserved_space_MB
,    SUM(data_space_MB)            as total_node_distribution_data_space_MB
,    SUM(index_space_MB)           as total_node_distribution_index_space_MB
,    SUM(unused_space_MB)          as total_node_distribution_unused_space_MB
FROM dbo.vTableSizes
GROUP BY     distribution_id
ORDER BY    distribution_id
;
```

Next steps

After creating the tables for your data warehouse, the next step is to load data into the table. For a loading tutorial, see Loading data from Azure blob storage with PolyBase.

Create Table As Select (CTAS) in SQL Data Warehouse

12/6/2017 • 10 min to read • Edit Online

Create table as select or `CTAS` is one of the most important T-SQL features available. It is a fully parallelized operation that creates a new table based on the output of a SELECT statement. `CTAS` is the simplest and fastest way to create a copy of a table. This document provides both examples and best practices for `CTAS`.

SELECT..INTO vs. CTAS

You can consider `CTAS` as a super-charged version of `SELECT..INTO`.

Below is an example of a simple `SELECT..INTO` statement:

```
SELECT *
INTO    [dbo].[FactInternetSales_new]
FROM    [dbo].[FactInternetSales]
```

In the example above `[dbo].[FactInternetSales_new]` would be created as ROUND_ROBIN distributed table with a CLUSTERED COLUMNSTORE INDEX on it as these are the table defaults in Azure SQL Data Warehouse.

`SELECT..INTO` however does not allow you to change either the distribution method or the index type as part of the operation. This is where `CTAS` comes in.

To convert the above to `CTAS` is quite straight-forward:

```
CREATE TABLE [dbo].[FactInternetSales_new]
WITH
(
    DISTRIBUTION = ROUND_ROBIN
,   CLUSTERED COLUMNSTORE INDEX
)
AS
SELECT  *
FROM    [dbo].[FactInternetSales]
;
```

With `CTAS` you are able to change both the distribution of the table data as well as the table type.

> **NOTE**
>
> If you are only trying to change the index in your `CTAS` operation and the source table is hash distributed then your `CTAS` operation will perform best if you maintain the same distribution column and data type. This will avoid cross distribution data movement during the operation which is more efficient.

Using CTAS to copy a table

Perhaps one of the most common uses of `CTAS` is creating a copy of a table so that you can change the DDL. If for example you originally created your table as `ROUND_ROBIN` and now want change it to a table distributed on a column, `CTAS` is how you would change the distribution column. `CTAS` can also be used to change partitioning,

Let's say you created this table using the default distribution type of `ROUND_ROBIN` distributed since no distribution column was specified in the `CREATE TABLE`.

```
CREATE TABLE 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) 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),
    CustomerPONumber nvarchar(25)
);
```

Now you want to create a new copy of this table with a Clustered Columnstore Index so that you can take advantage of the performance of Clustered Columnstore tables. You also want to distribute this table on ProductKey since you are anticipating joins on this column and want to avoid data movement during joins on ProductKey. Lastly you also want to add partitioning on OrderDateKey so that you can quickly delete old data by dropping old partitions. Here is the CTAS statement which would copy your old table into a new table.

```
CREATE TABLE FactInternetSales_new
WITH
(
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = HASH(ProductKey),
    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 FactInternetSales;
```

Finally you can rename your tables to swap in your new table and then drop your old table.

```
RENAME OBJECT FactInternetSales TO FactInternetSales_old;
RENAME OBJECT FactInternetSales_new TO FactInternetSales;

DROP TABLE FactInternetSales_old;
```

NOTE

Azure SQL Data Warehouse does not yet support auto create or auto update statistics. In order to get the best performance from your queries, it's important that statistics be created on all columns of all tables after the first load or any substantial changes occur in the data. For a detailed explanation of statistics, see the Statistics topic in the Develop group of topics.

Using CTAS to work around unsupported features

`CTAS` can also be used to work around a number of the unsupported features listed below. This can often prove to be a win/win situation as not only will your code be compliant but it will often execute faster on SQL Data Warehouse. This is as a result of its fully parallelized design. Scenarios that can be worked around with CTAS include:

- ANSI JOINS on UPDATEs
- ANSI JOINs on DELETEs
- MERGE statement

NOTE

Try to think "CTAS first". If you think you can solve a problem using `CTAS` then that is generally the best way to approach it - even if you are writing more data as a result.

ANSI join replacement for update statements

You may find you have a complex update that joins more than two tables together using ANSI joining syntax to perform the UPDATE or DELETE.

Imagine you had to update this table:

```
CREATE TABLE [dbo].[AnnualCategorySales]
(    [EnglishProductCategoryName]    NVARCHAR(50)    NOT NULL
,    [CalendarYear]                    SMALLINT        NOT NULL
,    [TotalSalesAmount]                MONEY            NOT NULL
)
WITH
(
    DISTRIBUTION = ROUND_ROBIN
)
;
```

The original query might have looked something like this:

```
UPDATE    acs
SET        [TotalSalesAmount] = [fis].[TotalSalesAmount]
FROM    [dbo].[AnnualCategorySales]     AS acs
JOIN    (
        SELECT    [EnglishProductCategoryName]
        ,        [CalendarYear]
        ,        SUM([SalesAmount])               AS [TotalSalesAmount]
        FROM    [dbo].[FactInternetSales]       AS s
        JOIN    [dbo].[DimDate]                   AS d    ON s.[OrderDateKey]               = d.[DateKey]
        JOIN    [dbo].[DimProduct]                AS p    ON s.[ProductKey]                = p.[ProductKey]
        JOIN    [dbo].[DimProductSubCategory]    AS u    ON p.[ProductSubcategoryKey]    = u.
[ProductSubcategoryKey]
        JOIN    [dbo].[DimProductCategory]        AS c    ON u.[ProductCategoryKey]        = c.
[ProductCategoryKey]
        WHERE     [CalendarYear] = 2004
        GROUP BY
                [EnglishProductCategoryName]
        ,        [CalendarYear]
        ) AS fis
ON    [acs].[EnglishProductCategoryName]    = [fis].[EnglishProductCategoryName]
AND    [acs].[CalendarYear]               = [fis].[CalendarYear]
;
```

Since SQL Data Warehouse does not support ANSI joins in the `FROM` clause of an `UPDATE` statement, you cannot copy this code over without changing it slightly.

You can use a combination of a `CTAS` and an implicit join to replace this code:

```
-- Create an interim table
CREATE TABLE CTAS_acs
WITH (DISTRIBUTION = ROUND_ROBIN)
AS
SELECT    ISNULL(CAST([EnglishProductCategoryName] AS NVARCHAR(50)),0)    AS [EnglishProductCategoryName]
,        ISNULL(CAST([CalendarYear] AS SMALLINT),0)                        AS [CalendarYear]
,        ISNULL(CAST(SUM([SalesAmount]) AS MONEY),0)                       AS [TotalSalesAmount]
FROM    [dbo].[FactInternetSales]        AS s
JOIN    [dbo].[DimDate]                    AS d    ON s.[OrderDateKey]               = d.[DateKey]
JOIN    [dbo].[DimProduct]                AS p    ON s.[ProductKey]                = p.[ProductKey]
JOIN    [dbo].[DimProductSubCategory]    AS u    ON p.[ProductSubcategoryKey]    = u.[ProductSubcategoryKey]
JOIN    [dbo].[DimProductCategory]        AS c    ON u.[ProductCategoryKey]        = c.[ProductCategoryKey]
WHERE     [CalendarYear] = 2004
GROUP BY
        [EnglishProductCategoryName]
,        [CalendarYear]
;

-- Use an implicit join to perform the update
UPDATE  AnnualCategorySales
SET     AnnualCategorySales.TotalSalesAmount = CTAS_ACS.TotalSalesAmount
FROM    CTAS_acs
WHERE   CTAS_acs.[EnglishProductCategoryName] = AnnualCategorySales.[EnglishProductCategoryName]
AND     CTAS_acs.[CalendarYear]               = AnnualCategorySales.[CalendarYear]
;

--Drop the interim table
DROP TABLE CTAS_acs
;
```

ANSI join replacement for delete statements

Sometimes the best approach for deleting data is to use `CTAS`. Rather than deleting the data simply select the data you want to keep. This especially true for `DELETE` statements that use ansi joining syntax since SQL Data

An example of a converted DELETE statement is available below:

```
CREATE TABLE dbo.DimProduct_upsert
WITH
(   Distribution=HASH(ProductKey)
,   CLUSTERED INDEX (ProductKey)
)
AS -- Select Data you wish to keep
SELECT     p.ProductKey
,          p.EnglishProductName
,          p.Color
FROM       dbo.DimProduct p
RIGHT JOIN dbo.stg_DimProduct s
ON         p.ProductKey = s.ProductKey
;

RENAME OBJECT dbo.DimProduct        TO DimProduct_old;
RENAME OBJECT dbo.DimProduct_upsert TO DimProduct;
```

Replace merge statements

Merge statements can be replaced, at least in part, by using `CTAS`. You can consolidate the `INSERT` and the `UPDATE` into a single statement. Any deleted records would need to be closed off in a second statement.

An example of an `UPSERT` is available below:

```
CREATE TABLE dbo.[DimProduct_upsert]
WITH
(   DISTRIBUTION = HASH([ProductKey])
,   CLUSTERED INDEX ([ProductKey])
)
AS
-- New rows and new versions of rows
SELECT      s.[ProductKey]
,           s.[EnglishProductName]
,           s.[Color]
FROM      dbo.[stg_DimProduct] AS s
UNION ALL
-- Keep rows that are not being touched
SELECT      p.[ProductKey]
,           p.[EnglishProductName]
,           p.[Color]
FROM      dbo.[DimProduct] AS p
WHERE NOT EXISTS
(   SELECT  *
    FROM    [dbo].[stg_DimProduct] s
    WHERE   s.[ProductKey] = p.[ProductKey]
)
;

RENAME OBJECT dbo.[DimProduct]          TO [DimProduct_old];
RENAME OBJECT dbo.[DimpProduct_upsert]  TO [DimProduct];
```

CTAS recommendation: Explicitly state data type and nullability of output

When migrating code you might find you run across this type of coding pattern:

```
DECLARE @d decimal(7,2) = 85.455
,       @f float(24)    = 85.455

CREATE TABLE result
(result DECIMAL(7,2) NOT NULL
)
WITH (DISTRIBUTION = ROUND_ROBIN)

INSERT INTO result
SELECT @d*@f
;
```

Instinctively you might think you should migrate this code to a CTAS and you would be correct. However, there is a hidden issue here.

The following code does NOT yield the same result:

```
DECLARE @d decimal(7,2) = 85.455
,       @f float(24)    = 85.455
;

CREATE TABLE ctas_r
WITH (DISTRIBUTION = ROUND_ROBIN)
AS
SELECT @d*@f as result
;
```

Notice that the column "result" carries forward the data type and nullability values of the expression. This can lead to subtle variances in values if you aren't careful.

Try the following as an example:

```
SELECT result,result*@d
from result
;

SELECT result,result*@d
from ctas_r
;
```

The value stored for result is different. As the persisted value in the result column is used in other expressions the error becomes even more significant.

	result	second_result
1	7302.98	624112.6708

	result	second_result
1	7302.984	624113.1

This is particularly important for data migrations. Even though the second query is arguably more accurate there is a problem. The data would be different compared to the source system and that leads to questions of integrity in the migration. This is one of those rare cases where the "wrong" answer is actually the right one!

The reason we see this disparity between the two results is down to implicit type casting. In the first example the table defines the column definition. When the row is inserted an implicit type conversion occurs. In the second example there is no implicit type conversion as the expression defines data type of the column. Notice also that the column in the second example has been defined as a NULLable column whereas in the first example it has not

was just left to the expression and by default this would result in a NULL definition.

To resolve these issues you must explicitly set the type conversion and nullability in the `SELECT` portion of the `CTAS` statement. You cannot set these properties in the create table part.

The example below demonstrates how to fix the code:

```
DECLARE @d decimal(7,2) = 85.455
,       @f float(24)    = 85.455

CREATE TABLE ctas_r
WITH (DISTRIBUTION = ROUND_ROBIN)
AS
SELECT ISNULL(CAST(@d*@f AS DECIMAL(7,2)),0) as result
```

Note the following:

- CAST or CONVERT could have been used
- ISNULL is used to force NULLability not COALESCE
- ISNULL is the outermost function
- The second part of the ISNULL is a constant i.e. 0

> **NOTE**
>
> For the nullability to be correctly set it is vital to use `ISNULL` and not `COALESCE`. `COALESCE` is not a deterministic function and so the result of the expression will always be NULLable. `ISNULL` is different. It is deterministic. Therefore when the second part of the `ISNULL` function is a constant or a literal then the resulting value will be NOT NULL.

This tip is not just useful for ensuring the integrity of your calculations. It is also important for table partition switching. Imagine you have this table defined as your fact:

```
CREATE TABLE [dbo].[Sales]
(
    [date]      INT     NOT NULL
,   [product]   INT     NOT NULL
,   [store]     INT     NOT NULL
,   [quantity]  INT     NOT NULL
,   [price]     MONEY   NOT NULL
,   [amount]    MONEY   NOT NULL
)
WITH
(   DISTRIBUTION = HASH([product])
,   PARTITION   (   [date] RANGE RIGHT FOR VALUES
                    (20000101,20010101,20020101
                    ,20030101,20040101,20050101
                    )
                )
)
;
```

However, the value field is a calculated expression it is not part of the source data.

To create your partitioned dataset you might want to do this:

```
CREATE TABLE [dbo].[Sales_in]
WITH
(   DISTRIBUTION = HASH([product])
,   PARTITION   (   [date] RANGE RIGHT FOR VALUES
                    (20000101,20010101
                    )
                )
)
AS
SELECT
    [date]
,   [product]
,   [store]
,   [quantity]
,   [price]
,   [quantity]*[price]  AS [amount]
FROM [stg].[source]
OPTION (LABEL = 'CTAS : Partition IN table : Create')
;
```

The query would run perfectly fine. The problem comes when you try to perform the partition switch. The table definitions do not match. To make the table definitions match the CTAS needs to be modified.

```
CREATE TABLE [dbo].[Sales_in]
WITH
(   DISTRIBUTION = HASH([product])
,   PARTITION   (   [date] RANGE RIGHT FOR VALUES
                    (20000101,20010101
                    )
                )
)
AS
SELECT
    [date]
,   [product]
,   [store]
,   [quantity]
,   [price]
,   ISNULL(CAST([quantity]*[price] AS MONEY),0) AS [amount]
FROM [stg].[source]
OPTION (LABEL = 'CTAS : Partition IN table : Create');
```

You can see therefore that type consistency and maintaining nullability properties on a CTAS is a good engineering best practice. It helps to maintain integrity in your calculations and also ensures that partition switching is possible.

Please refer to MSDN for more information on using CTAS. It is one of the most important statements in Azure SQL Data Warehouse. Make sure you thoroughly understand it.

Next steps

For more development tips, see development overview.

Guidance for defining data types for tables in SQL Data Warehouse

12/6/2017 • 2 min to read • Edit Online

Use these recommendations to define table data types that are compatible with SQL Data Warehouse. In addition to compatibility, minimizing the size of data types improves query performance.

SQL Data Warehouse supports the most commonly used data types. For a list of the supported data types, see data types in the CREATE TABLE statement.

Minimize row length

Minimizing the size of data types shortens the row length, which leads to better query performance. Use the smallest data type that works for your data.

- Avoid defining character columns with a large default length. For example, if the longest value is 25 characters, then define your column as VARCHAR(25).
- Avoid using NVARCHAR when you only need VARCHAR.
- When possible, use NVARCHAR(4000) or VARCHAR(8000) instead of NVARCHAR(MAX) or VARCHAR(MAX).

If you are using Polybase to load your tables, the defined length of the table row cannot exceed 1 MB. When a row with variable-length data exceeds 1 MB, you can load the row with BCP, but not with PolyBase.

Identify unsupported data types

If you are migrating your database from another SQL database, you might encounter data types that are not supported in SQL Data Warehouse. Use this query to discover unsupported data types in your existing SQL schema.

```
SELECT  t.[name], c.[name], c.[system_type_id], c.[user_type_id], y.[is_user_defined], y.[name]
FROM sys.tables  t
JOIN sys.columns c on t.[object_id]    = c.[object_id]
JOIN sys.types   y on c.[user_type_id] = y.[user_type_id]
WHERE y.[name] IN
('geography','geometry','hierarchyid','image','text','ntext','sql_variant','timestamp','xml')
 AND  y.[is_user_defined] = 1;
```

Use workarounds for unsupported data types

The following list shows the data types that SQL Data Warehouse does not support and gives alternatives that you can use instead of the unsupported data types.

UNSUPPORTED DATA TYPE	WORKAROUND
geometry	varbinary
geography	varbinary
hierarchyid	nvarchar(4000)

UNSUPPORTED DATA TYPE	WORKAROUND
image	varbinary
text	varchar
ntext	nvarchar
sql_variant	Split column into several strongly typed columns.
table	Convert to temporary tables.
timestamp	Rework code to use datetime2 and `CURRENT_TIMESTAMP` function. Only constants are supported as defaults, therefore current_timestamp cannot be defined as a default constraint. If you need to migrate row version values from a timestamp typed column, then use BINARY(8) or VARBINARY(8) for NOT NULL or NULL row version values.
xml	varchar
user-defined type	Convert back to the native data type when possible.
default values	Default values support literals and constants only. Non-deterministic expressions or functions, such as `GETDATE()` or `CURRENT_TIMESTAMP`, are not supported.

Next steps

To learn more, see:

- SQL Data Warehouse Best Practices
- Table Overview
- Distributing a Table
- Indexing a Table
- Partitioning a Table
- Maintaining Table Statistics
- Temporary Tables

Distributing tables in SQL Data Warehouse

12/6/2017 • 13 min to read • Edit Online

SQL Data Warehouse is a massively parallel processing (MPP) distributed database system. By dividing data and processing capability across multiple nodes, SQL Data Warehouse can offer huge scalability - far beyond any single system. Deciding how to distribute your data within your SQL Data Warehouse is one of the most important factors to achieving optimal performance. The key to optimal performance is minimizing data movement and in turn the key to minimizing data movement is selecting the right distribution strategy.

Understanding data movement

In an MPP system, the data from each table is divided across several underlying databases. The most optimized queries on an MPP system can simply be passed through to execute on the individual distributed databases with no interaction between the other databases. For example, let's say you have a database with sales data which contains two tables, sales and customers. If you have a query that needs to join your sales table to your customer table and you divide both your sales and customer tables up by customer number, putting each customer in a separate database, any queries which join sales and customer can be solved within each database with no knowledge of the other databases. In contrast, if you divided your sales data by order number and your customer data by customer number, then any given database will not have the corresponding data for each customer and thus if you wanted to join your sales data to your customer data, you would need to get the data for each customer from the other databases. In this second example, data movement would need to occur to move the customer data to the sales data, so that the two tables can be joined.

Data movement isn't always a bad thing, sometimes it's necessary to solve a query. But when this extra step can be avoided, naturally your query will run faster. Data Movement most commonly arises when tables are joined or aggregations are performed. Often you need to do both, so while you may be able to optimize for one scenario, like a join, you still need data movement to help you solve for the other scenario, like an aggregation. The trick is figuring out which is less work. In most cases, distributing large fact tables on a commonly joined column is the most effective method for reducing the most data movement. Distributing data on join columns is a much more common method to reduce data movement than distributing data on columns involved in an aggregation.

Select distribution method

Behind the scenes, SQL Data Warehouse divides your data into 60 databases. Each individual database is referred to as a **distribution**. When data is loaded into each table, SQL Data Warehouse has to know how to divide your data across these 60 distributions.

The distribution method is defined at the table level and currently there are two choices:

1. **Round robin** which distribute data evenly but randomly.
2. **Hash Distributed** which distributes data based on hashing values from a single column

By default, when you do not define a data distribution method, your table will be distributed using the **round robin** distribution method. However, as you become more sophisticated in your implementation, you will want to consider using **hash distributed** tables to minimize data movement which will in turn optimize query performance.

Round Robin Tables

Using the Round Robin method of distributing data is very much how it sounds. As your data is loaded, each

process which places your data. A round robin distribution is sometimes called a random hash for this reason. With a round-robin distributed table there is no need to understand the data. For this reason, Round-Robin tables often make good loading targets.

By default, if no distribution method is chosen, the round robin distribution method will be used. However, while round robin tables are easy to use, because data is randomly distributed across the system it means that the system can't guarantee which distribution each row is on. As a result, the system sometimes needs to invoke a data movement operation to better organize your data before it can resolve a query. This extra step can slow down your queries.

Consider using Round Robin distribution for your table in the following scenarios:

- When getting started as a simple starting point
- If there is no obvious joining key
- If there is not good candidate column for hash distributing the table
- If the table does not share a common join key with other tables
- If the join is less significant than other joins in the query
- When the table is a temporary staging table

Both of these examples will create a Round Robin Table:

```
-- Round Robin created by default
CREATE TABLE [dbo].[FactInternetSales]
(   [ProductKey]            int          NOT NULL
,   [OrderDateKey]          int          NOT NULL
,   [CustomerKey]           int          NOT NULL
,   [PromotionKey]          int          NOT NULL
,   [SalesOrderNumber]      nvarchar(20) NOT NULL
,   [OrderQuantity]         smallint     NOT NULL
,   [UnitPrice]             money        NOT NULL
,   [SalesAmount]           money        NOT NULL
)
;

-- Explicitly Created Round Robin Table
CREATE TABLE [dbo].[FactInternetSales]
(   [ProductKey]            int          NOT NULL
,   [OrderDateKey]          int          NOT NULL
,   [CustomerKey]           int          NOT NULL
,   [PromotionKey]          int          NOT NULL
,   [SalesOrderNumber]      nvarchar(20) NOT NULL
,   [OrderQuantity]         smallint     NOT NULL
,   [UnitPrice]             money        NOT NULL
,   [SalesAmount]           money        NOT NULL
)
WITH
(   CLUSTERED COLUMNSTORE INDEX
,   DISTRIBUTION = ROUND_ROBIN
)
;
```

> **NOTE**
> While round robin is the default table type being explicit in your DDL is considered a best practice so that the intentions of your table layout are clear to others.

Hash Distributed Tables

distributed databases using a hashing algorithm on a single column which you select. The distribution column is what determines how the data is divided across your distributed databases. The hash function uses the distribution column to assign rows to distributions. The hashing algorithm and resulting distribution is deterministic. That is the same value with the same data type will always has to the same distribution.

This example will create a table distributed on id:

```
CREATE TABLE [dbo].[FactInternetSales]
(   [ProductKey]            int          NOT NULL
,   [OrderDateKey]          int          NOT NULL
,   [CustomerKey]           int          NOT NULL
,   [PromotionKey]          int          NOT NULL
,   [SalesOrderNumber]      nvarchar(20) NOT NULL
,   [OrderQuantity]         smallint     NOT NULL
,   [UnitPrice]             money        NOT NULL
,   [SalesAmount]           money        NOT NULL
)
WITH
(   CLUSTERED COLUMNSTORE INDEX
,  DISTRIBUTION = HASH([ProductKey])
)
;
```

Select distribution column

When you choose to **hash distribute** a table, you will need to select a single distribution column. When selecting a distribution column, there are three major factors to consider.

Select a single column which will:

1. Not be updated
2. Distribute data evenly, avoiding data skew
3. Minimize data movement

Select distribution column which will not be updated

Distribution columns are not updatable, therefore, select a column with static values. If a column will need to be updated, it is generally not a good distribution candidate. If there is a case where you must update a distribution column, this can be done by first deleting the row and then inserting a new row.

Select distribution column which will distribute data evenly

Since a distributed system performs only as fast as its slowest distribution, it is important to divide the work evenly across the distributions in order to achieve balanced execution across the system. The way the work is divided on a distributed system is based on where the data for each distribution lives. This makes it very important to select the right distribution column for distributing the data so that each distribution has equal work and will take the same time to complete its portion of the work. When work is well divided across the system, the data is balanced across the distributions. When data is not evenly balanced, we call this **data skew**.

To divide data evenly and avoid data skew, consider the following when selecting your distribution column:

1. Select a column which contains a significant number of distinct values.
2. Avoid distributing data on columns with a few distinct values.
3. Avoid distributing data on columns with a high frequency of nulls.
4. Avoid distributing data on date columns.

Since each value is hashed to 1 of 60 distributions, to achieve even distribution you will want to select a column that is highly unique and contains more than 60 unique values. To illustrate, consider a case where a column

on 40 distributions at most, leaving 20 distributions with no data and no processing to do. Conversely, the other 40 distributions would have more work to do that if the data was evenly spread over 60 distributions. This scenario is an example of data skew.

In MPP system, each query step waits for all distributions to complete their share of the work. If one distribution is doing more work than the others, then the resource of the other distributions are essentially wasted just waiting on the busy distribution. When work is not evenly spread across all distributions, we call this **processing skew**. Processing skew will cause queries to run slower than if the workload can be evenly spread across the distributions. Data skew will lead to processing skew.

Avoid distributing on highly nullable column as the null values will all land on the same distribution. Distributing on a date column can also cause processing skew because all data for a given date will land on the same distribution. If several users are executing queries all filtering on the same date, then only 1 of the 60 distributions will be doing all of the work since a given date will only be on one distribution. In this scenario, the queries will likely run 60 times slower than if the data were equally spread over all of the distributions.

When no good candidate columns exist, then consider using round robin as the distribution method.

Select distribution column which will minimize data movement

Minimizing data movement by selecting the right distribution column is one of the most important strategies for optimizing performance of your SQL Data Warehouse. Data Movement most commonly arises when tables are joined or aggregations are performed. Columns used in `JOIN`, `GROUP BY`, `DISTINCT`, `OVER` and `HAVING` clauses all make for **good** hash distribution candidates.

On the other hand, columns in the `WHERE` clause do **not** make for good hash column candidates because they limit which distributions participate in the query, causing processing skew. A good example of a column which might be tempting to distribute on, but often can cause this processing skew is a date column.

Generally speaking, if you have two large fact tables frequently involved in a join, you will gain the most performance by distributing both tables on one of the join columns. If you have a table that is never joined to another large fact table, then look to columns that are frequently in the `GROUP BY` clause.

There are a few key criteria which must be met to avoid data movement during a join:

1. The tables involved in the join must be hash distributed on **one** of the columns participating in the join.
2. The data types of the join columns must match between both tables.
3. The columns must be joined with an equals operator.
4. The join type may not be a `CROSS JOIN`.

Troubleshooting data skew

When table data is distributed using the hash distribution method there is a chance that some distributions will be skewed to have disproportionately more data than others. Excessive data skew can impact query performance because the final result of a distributed query must wait for the longest running distribution to finish. Depending on the degree of the data skew you might need to address it.

Identifying skew

A simple way to identify a table as skewed is to use `DBCC PDW_SHOWSPACEUSED`. This is a very quick and simple way to see the number of table rows that are stored in each of the 60 distributions of your database. Remember that for the most balanced performance, the rows in your distributed table should be spread evenly across all the distributions.

```
-- Find data skew for a distributed table
DBCC PDW_SHOWSPACEUSED('dbo.FactInternetSales');
```

However, if you query the Azure SQL Data Warehouse dynamic management views (DMV) you can perform a more detailed analysis. To start, create the view dbo.vTableSizes view using the SQL from Table Overview article. Once the view is created, run this query to identify which tables have more than 10% data skew.

```
select *
from dbo.vTableSizes
where two_part_name in
    (
    select two_part_name
    from dbo.vTableSizes
    where row_count > 0
    group by two_part_name
    having min(row_count * 1.000)/max(row_count * 1.000) > .10
    )
order by two_part_name, row_count
;
```

Resolving data skew

Not all skew is enough to warrant a fix. In some cases, the performance of a table in some queries can outweigh the harm of data skew. To decide if you should resolve data skew in a table, you should understand as much as possible about the data volumes and queries in your workload. One way to look at the impact of skew is to use the steps in the Query Monitoring article to monitor the impact of skew on query performance and specifically the impact to how long queries take to complete on the individual distributions.

Distributing data is a matter of finding the right balance between minimizing data skew and minimizing data movement. These can be opposing goals, and sometimes you will want to keep data skew in order to reduce data movement. For example, when the distribution column is frequently the shared column in joins and aggregations, you will be minimizing data movement. The benefit of having the minimal data movement might outweigh the impact of having data skew.

The typical way to resolve data skew is to re-create the table with a different distribution column. Since there is no way to change the distribution column on an existing table, the way to change the distribution of a table it to recreate it with a [CTAS][]. Here are two examples of how resolve data skew:

Example 1: Re-create the table with a new distribution column

This example uses [CTAS][] to re-create a table with a different hash distribution column.

```
CREATE TABLE [dbo].[FactInternetSales_CustomerKey]
WITH (  CLUSTERED COLUMNSTORE INDEX
     ,  DISTRIBUTION =  HASH([CustomerKey])
     ,  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    [dbo].[FactInternetSales]
OPTION  (LABEL  = 'CTAS : FactInternetSales_CustomerKey')
;

--Create statistics on new table
CREATE STATISTICS [ProductKey] ON [FactInternetSales_CustomerKey] ([ProductKey]);
CREATE STATISTICS [OrderDateKey] ON [FactInternetSales_CustomerKey] ([OrderDateKey]);
CREATE STATISTICS [CustomerKey] ON [FactInternetSales_CustomerKey] ([CustomerKey]);
CREATE STATISTICS [PromotionKey] ON [FactInternetSales_CustomerKey] ([PromotionKey]);
CREATE STATISTICS [SalesOrderNumber] ON [FactInternetSales_CustomerKey] ([SalesOrderNumber]);
CREATE STATISTICS [OrderQuantity] ON [FactInternetSales_CustomerKey] ([OrderQuantity]);
CREATE STATISTICS [UnitPrice] ON [FactInternetSales_CustomerKey] ([UnitPrice]);
CREATE STATISTICS [SalesAmount] ON [FactInternetSales_CustomerKey] ([SalesAmount]);

--Rename the tables
RENAME OBJECT [dbo].[FactInternetSales] TO [FactInternetSales_ProductKey];
RENAME OBJECT [dbo].[FactInternetSales_CustomerKey] TO [FactInternetSales];
```

Example 2: Re-create the table using round robin distribution

This example uses [CTAS][] to re-create a table with round robin instead of a hash distribution. This change will produce even data distribution at the cost of increased data movement.

```
CREATE TABLE [dbo].[FactInternetSales_ROUND_ROBIN]
WITH (  CLUSTERED COLUMNSTORE INDEX
     ,  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    [dbo].[FactInternetSales]
OPTION  (LABEL  = 'CTAS : FactInternetSales_ROUND_ROBIN')
;

--Create statistics on new table
CREATE STATISTICS [ProductKey] ON [FactInternetSales_ROUND_ROBIN] ([ProductKey]);
CREATE STATISTICS [OrderDateKey] ON [FactInternetSales_ROUND_ROBIN] ([OrderDateKey]);
CREATE STATISTICS [CustomerKey] ON [FactInternetSales_ROUND_ROBIN] ([CustomerKey]);
CREATE STATISTICS [PromotionKey] ON [FactInternetSales_ROUND_ROBIN] ([PromotionKey]);
CREATE STATISTICS [SalesOrderNumber] ON [FactInternetSales_ROUND_ROBIN] ([SalesOrderNumber]);
CREATE STATISTICS [OrderQuantity] ON [FactInternetSales_ROUND_ROBIN] ([OrderQuantity]);
CREATE STATISTICS [UnitPrice] ON [FactInternetSales_ROUND_ROBIN] ([UnitPrice]);
CREATE STATISTICS [SalesAmount] ON [FactInternetSales_ROUND_ROBIN] ([SalesAmount]);

--Rename the tables
RENAME OBJECT [dbo].[FactInternetSales] TO [FactInternetSales_HASH];
RENAME OBJECT [dbo].[FactInternetSales_ROUND_ROBIN] TO [FactInternetSales];
```

Next steps

To learn more about table design, see the Distribute, Index, Partition, Data Types, Statistics and Temporary Tables articles.

For an overview of best practices, see SQL Data Warehouse Best Practices.

Indexing tables in SQL Data Warehouse

12/6/2017 • 15 min to read • Edit Online

SQL Data Warehouse offers several indexing options including clustered columnstore indexes, clustered indexes and nonclustered indexes. In addition, it also offers a no index option also known as heap. This article covers the benefits of each index type as well as tips to getting the most performance out of your indexes. See create table syntax for more detail on how to create a table in SQL Data Warehouse.

Clustered columnstore indexes

By default, SQL Data Warehouse creates a clustered columnstore index when no index options are specified on a table. Clustered columnstore tables offer both the highest level of data compression as well as the best overall query performance. Clustered columnstore tables will generally outperform clustered index or heap tables and are usually the best choice for large tables. For these reasons, clustered columnstore is the best place to start when you are unsure of how to index your table.

To create a clustered columnstore table, simply specify CLUSTERED COLUMNSTORE INDEX in the WITH clause, or leave the WITH clause off:

```
CREATE TABLE myTable
  (
    id int NOT NULL,
    lastName varchar(20),
    zipCode varchar(6)
  )
WITH ( CLUSTERED COLUMNSTORE INDEX );
```

There are a few scenarios where clustered columnstore may not be a good option:

- Columnstore tables do not support varchar(max), nvarchar(max) and varbinary(max). Consider heap or clustered index instead.
- Columnstore tables may be less efficient for transient data. Consider heap and perhaps even temporary tables.
- Small tables with less than 100 million rows. Consider heap tables.

Heap tables

When you are temporarily landing data on SQL Data Warehouse, you may find that using a heap table will make the overall process faster. This is because loads to heaps are faster than to index tables and in some cases the subsequent read can be done from cache. 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 temporary table will also load much faster than loading a table to permanent storage.

For small lookup tables, less than 100 million rows, often heap tables make sense. Cluster columnstore tables begin to achieve optimal compression once there is more than 100 million rows.

To create a heap table, simply specify HEAP in the WITH clause:

```
CREATE TABLE myTable
  (
    id int NOT NULL,
    lastName varchar(20),
    zipCode varchar(6)
  )
WITH ( HEAP );
```

Clustered and nonclustered indexes

Clustered indexes may outperform clustered columnstore tables when a single row needs to be quickly retrieved. For queries where a single or very few row lookup is required to performance with extreme speed, consider a cluster index or nonclustered secondary index. The disadvantage to using a clustered index is that only queries which use a highly selective filter on the clustered index column will benefit. To improve filter on other columns a nonclustered index can be added to other columns. However, each index which is added to a table will add both space and processing time to loads.

To create a clustered index table, simply specify CLUSTERED INDEX in the WITH clause:

```
CREATE TABLE myTable
  (
    id int NOT NULL,
    lastName varchar(20),
    zipCode varchar(6)
  )
WITH ( CLUSTERED INDEX (id) );
```

To add a non-clustered index on a table, simply use the following syntax:

```
CREATE INDEX zipCodeIndex ON t1 (zipCode);
```

Optimizing clustered columnstore indexes

Clustered columnstore tables are organized in data into segments. Having high segment quality is critical to achieving optimal query performance on a columnstore table. Segment quality can be measured by the number of rows in a compressed row group. Segment quality is most optimal where there are at least 100K rows per compressed row group and gain in performance as the number of rows per row group approach 1,048,576 rows, which is the most rows a row group can contain.

The below view can be created and used on your system to compute the average rows per row group and identify any sub-optimal cluster columnstore indexes. The last column on this view will generate as SQL statement which can be used to rebuild your indexes.

```
CREATE VIEW dbo.vColumnstoreDensity
AS
SELECT
        GETDATE()                                                                   AS [execution_date]
,       DB_Name()                                                                   AS [database_name]
,       s.name                                                                      AS [schema_name]
,       t.name                                                                      AS [table_name]
,    COUNT(DISTINCT rg.[partition_number])                    AS [table_partition_count]
,       SUM(rg.[total_rows])                                                        AS [row_count_total]
,       SUM(rg.[total_rows])/COUNT(DISTINCT rg.[distribution_id])                   AS
[row_count_per_distribution_MAX]
,    CEILING    ((SUM(rg.[total_rows])*1.0/COUNT(DISTINCT rg.[distribution_id]))/1048576) AS
[rowgroup_per_distribution_MAX]
,       SUM(CASE WHEN rg.[State] = 0 THEN 1                   ELSE 0    END)    AS
[INVISIBLE_rowgroup_count]
,       SUM(CASE WHEN rg.[State] = 0 THEN rg.[total_rows]     ELSE 0    END)    AS [INVISIBLE_rowgroup_rows]
,       MIN(CASE WHEN rg.[State] = 0 THEN rg.[total_rows]     ELSE NULL END)    AS
[INVISIBLE_rowgroup_rows_MIN]
,       MAX(CASE WHEN rg.[State] = 0 THEN rg.[total_rows]     ELSE NULL END)    AS
[INVISIBLE_rowgroup_rows_MAX]
,       AVG(CASE WHEN rg.[State] = 0 THEN rg.[total_rows]     ELSE NULL END)    AS
[INVISIBLE_rowgroup_rows_AVG]
,       SUM(CASE WHEN rg.[State] = 1 THEN 1                   ELSE 0    END)    AS [OPEN_rowgroup_count]
,       SUM(CASE WHEN rg.[State] = 1 THEN rg.[total_rows]     ELSE 0    END)    AS [OPEN_rowgroup_rows]
,       MIN(CASE WHEN rg.[State] = 1 THEN rg.[total_rows]     ELSE NULL END)    AS [OPEN_rowgroup_rows_MIN]
,       MAX(CASE WHEN rg.[State] = 1 THEN rg.[total_rows]     ELSE NULL END)    AS [OPEN_rowgroup_rows_MAX]
,       AVG(CASE WHEN rg.[State] = 1 THEN rg.[total_rows]     ELSE NULL END)    AS [OPEN_rowgroup_rows_AVG]
,       SUM(CASE WHEN rg.[State] = 2 THEN 1                   ELSE 0    END)    AS [CLOSED_rowgroup_count]
,       SUM(CASE WHEN rg.[State] = 2 THEN rg.[total_rows]     ELSE 0    END)    AS [CLOSED_rowgroup_rows]
,       MIN(CASE WHEN rg.[State] = 2 THEN rg.[total_rows]     ELSE NULL END)    AS
[CLOSED_rowgroup_rows_MIN]
,       MAX(CASE WHEN rg.[State] = 2 THEN rg.[total_rows]     ELSE NULL END)    AS
[CLOSED_rowgroup_rows_MAX]
,       AVG(CASE WHEN rg.[State] = 2 THEN rg.[total_rows]     ELSE NULL END)    AS
[CLOSED_rowgroup_rows_AVG]
,       SUM(CASE WHEN rg.[State] = 3 THEN 1                   ELSE 0    END)    AS
[COMPRESSED_rowgroup_count]
,       SUM(CASE WHEN rg.[State] = 3 THEN rg.[total_rows]     ELSE 0    END)    AS
[COMPRESSED_rowgroup_rows]
,       SUM(CASE WHEN rg.[State] = 3 THEN rg.[deleted_rows]   ELSE 0    END)    AS
[COMPRESSED_rowgroup_rows_DELETED]
,       MIN(CASE WHEN rg.[State] = 3 THEN rg.[total_rows]     ELSE NULL END)    AS
[COMPRESSED_rowgroup_rows_MIN]
,       MAX(CASE WHEN rg.[State] = 3 THEN rg.[total_rows]     ELSE NULL END)    AS
[COMPRESSED_rowgroup_rows_MAX]
,       AVG(CASE WHEN rg.[State] = 3 THEN rg.[total_rows]     ELSE NULL END)    AS
[COMPRESSED_rowgroup_rows_AVG]
,       'ALTER INDEX ALL ON ' + s.name + '.' + t.NAME + ' REBUILD;'               AS [Rebuild_Index_SQL]
FROM    sys.[pdw_nodes_column_store_row_groups] rg
JOIN    sys.[pdw_nodes_tables] nt                   ON  rg.[object_id]          = nt.[object_id]
                                                    AND rg.[pdw_node_id]        = nt.[pdw_node_id]
                                                    AND rg.[distribution_id]    = nt.[distribution_id]
JOIN    sys.[pdw_table_mappings] mp                 ON  nt.[name]               = mp.[physical_name]
JOIN    sys.[tables] t                              ON  mp.[object_id]          = t.[object_id]
JOIN    sys.[schemas] s                             ON t.[schema_id]            = s.[schema_id]
GROUP BY
        s.[name]
,       t.[name]
;
```

Now that you have created the view, run this query to identify tables with row groups with less than 100K rows. Of course, you may want to increase the threshold of 100K if you are looking for more optimal segment quality.

```
SELECT    *
FROM    [dbo].[vColumnstoreDensity]
WHERE    COMPRESSED_rowgroup_rows_AVG < 100000
        OR INVISIBLE_rowgroup_rows_AVG < 100000
```

Once you have run the query you can begin to look at the data and analyze your results. This table explains what to look for in your row group analysis.

COLUMN	HOW TO USE THIS DATA
[table_partition_count]	If the table is partitioned, then you may expect to see higher Open row group counts. Each partition in the distribution could in theory have an open row group associated with it. Factor this into your analysis. A small table that has been partitioned could be optimized by removing the partitioning altogether as this would improve compression.
[row_count_total]	Total row count for the table. For example, you can use this value to calculate percentage of rows in the compressed state.
[row_count_per_distribution_MAX]	If all rows are evenly distributed this value would be the target number of rows per distribution. Compare this value with the compressed_rowgroup_count.
[COMPRESSED_rowgroup_rows]	Total number of rows in columnstore format for the table.
[COMPRESSED_rowgroup_rows_AVG]	If the average number of rows is significantly less than the maximum # of rows for a row group, then consider using CTAS or ALTER INDEX REBUILD to recompress the data
[COMPRESSED_rowgroup_count]	Number of row groups in columnstore format. If this number is very high in relation to the table it is an indicator that the columnstore density is low.
[COMPRESSED_rowgroup_rows_DELETED]	Rows are logically deleted in columnstore format. If the number is high relative to table size, consider recreating the partition or rebuilding the index as this removes them physically.
[COMPRESSED_rowgroup_rows_MIN]	Use this in conjunction with the AVG and MAX columns to understand the range of values for the row groups in your columnstore. A low number over the load threshold (102,400 per partition aligned distribution) suggests that optimizations are available in the data load
[COMPRESSED_rowgroup_rows_MAX]	As above
[OPEN_rowgroup_count]	Open row groups are normal. One would reasonably expect one OPEN row group per table distribution (60). Excessive numbers suggest data loading across partitions. Double check the partitioning strategy to make sure it is sound
[OPEN_rowgroup_rows]	Each row group can have 1,048,576 rows in it as a maximum. Use this value to see how full the open row groups are currently

COLUMN	HOW TO USE THIS DATA
[OPEN_rowgroup_rows_MIN]	Open groups indicate that data is either being trickle loaded into the table or that the previous load spilled over remaining rows into this row group. Use the MIN, MAX, AVG columns to see how much data is sat in OPEN row groups. For small tables it could be 100% of all the data! In which case ALTER INDEX REBUILD to force the data to columnstore.
[OPEN_rowgroup_rows_MAX]	As above
[OPEN_rowgroup_rows_AVG]	As above
[CLOSED_rowgroup_rows]	Look at the closed row group rows as a sanity check.
[CLOSED_rowgroup_count]	The number of closed row groups should be low if any are seen at all. Closed row groups can be converted to compressed row groups using the ALTER INDEX ... REORGANISE command. However, this is not normally required. Closed groups are automatically converted to columnstore row groups by the background "tuple mover" process.
[CLOSED_rowgroup_rows_MIN]	Closed row groups should have a very high fill rate. If the fill rate for a closed row group is low, then further analysis of the columnstore is required.
[CLOSED_rowgroup_rows_MAX]	As above
[CLOSED_rowgroup_rows_AVG]	As above
[Rebuild_Index_SQL]	SQL to rebuild columnstore index for a table

Causes of poor columnstore index quality

If you have identified tables with poor segment quality, you will want to identify the root cause. Below are some other common causes of poor segment quality:

1. Memory pressure when index was built
2. High volume of DML operations
3. Small or trickle load operations
4. Too many partitions

These factors can cause a columnstore index to have significantly less than the optimal 1 million rows per row group. They can also cause rows to go to the delta row group instead of a compressed row group.

Memory pressure when index was built

The number of rows per compressed row group are directly related to the width of the row and the amount of memory available to process the row group. When rows are written to columnstore tables under memory pressure, columnstore segment quality may suffer. Therefore, the best practice is to give the session which is writing to your columnstore index tables access to as much memory as possible. Since there is a trade-off between memory and concurrency, the guidance on the right memory allocation depends on the data in each row of your table, the data warehouse units allocated to your system, and the number of concurrency slots you can give to the session which is writing data to your table. As a best practice, we recommend starting with

using DW1000 and above.

High volume of DML operations

A high volume of DML operations that update and delete rows can introduce inefficiency into the columnstore. This is especially true when the majority of the rows in a row group are modified.

- Deleting a row from a compressed row group only logically marks the row as deleted. The row remains in the compressed row group until the partition or table is rebuilt.
- Inserting a row adds the row to to an internal rowstore table called a delta row group. The inserted row is not converted to columnstore until the delta row group is full and is marked as closed. Row groups are closed once they reach the maximum capacity of 1,048,576 rows.
- Updating a row in columnstore format is processed as a logical delete and then an insert. The inserted row may be stored in the delta store.

Batched update and insert operations that exceed the bulk threshold of 102,400 rows per partition aligned distribution will be written directly to the columnstore format. However, assuming an even distribution, you would need to be modifying more than 6.144 million rows in a single operation for this to occur. If the number of rows for a given partition aligned distribution is less than 102,400 then the rows will go to the delta store and will stay there until sufficient rows have been inserted or modified to close the row group or the index has been rebuilt.

Small or trickle load operations

Small loads that flow into SQL Data Warehouse are also sometimes known as trickle loads. They typically represent a near constant stream of data being ingested by the system. However, as this stream is near continuous the volume of rows is not particularly large. More often than not the data is significantly under the threshold required for a direct load to columnstore format.

In these situations, it is often better to land the data first in Azure blob storage and let it accumulate prior to loading. This technique is often known as *micro-batching*.

Too many partitions

Another thing to consider is the impact of partitioning on your clustered columnstore tables. Before partitioning, SQL Data Warehouse already divides your data into 60 databases. Partitioning further divides your data. If you partition your data, then you will want to consider that **each** partition will need to have at least 1 million rows to benefit from a clustered columnstore index. If you partition your table into 100 partitions, then your table will need to have at least 6 billion rows to benefit from a clustered columnstore index (60 distributions * 100 partitions * 1 million rows). If your 100 partition table does not have 6 billion rows, either reduce the number of partitions or consider using a heap table instead.

Once your tables have been loaded with some data, follow the below steps to identify and rebuild tables with sub-optimal cluster columnstore indexes.

Rebuilding indexes to improve segment quality

Step 1: Identify or create user which uses the right resource class

One quick way to immediately improve segment quality is to rebuild the index. The SQL returned by the above view will return an ALTER INDEX REBUILD statement which can be used to rebuild your indexes. When rebuilding your indexes, be sure that you allocate enough memory to the session which will rebuild your index. To do this, increase the resource class of a user which has permissions to rebuild the index on this table to the recommended minimum. The resource class of the database owner user cannot be changed, so if you have not created a user on the system, you will need to do so first. The minimum we recommend is xlargerc if you are using DW300 or less, largerc if you are using DW400 to DW600, and mediumrc if you are using DW1000 and above.

information about resource classes and how to create a new user can be found in the concurrency and workload management article.

```
EXEC sp_addrolemember 'xlargerc', 'LoadUser'
```

Step 2: Rebuild clustered columnstore indexes with higher resource class user

Logon as the user from step 1 (e.g. LoadUser), which is now using a higher resource class, and execute the ALTER INDEX statements. Be sure that this user has ALTER permission to the tables where the index is being rebuilt. These examples show how to rebuild the entire columnstore index or how to rebuild a single partition. On large tables, it is more practical to rebuild indexes a single partition at a time.

Alternatively, instead of rebuilding the index, you could copy the table to a new table using CTAS. Which way is best? For large volumes of data, CTAS is usually faster than ALTER INDEX. For smaller volumes of data, ALTER INDEX is easier to use and won't require you to swap out the table. See **Rebuilding indexes with CTAS and partition switching** below for more details on how to rebuild indexes with CTAS.

```
-- Rebuild the entire clustered index
ALTER INDEX ALL ON [dbo].[DimProduct] REBUILD
```

```
-- Rebuild a single partition
ALTER INDEX ALL ON [dbo].[FactInternetSales] REBUILD Partition = 5
```

```
-- Rebuild a single partition with archival compression
ALTER INDEX ALL ON [dbo].[FactInternetSales] REBUILD Partition = 5 WITH (DATA_COMPRESSION =
COLUMNSTORE_ARCHIVE)
```

```
-- Rebuild a single partition with columnstore compression
ALTER INDEX ALL ON [dbo].[FactInternetSales] REBUILD Partition = 5 WITH (DATA_COMPRESSION = COLUMNSTORE)
```

Rebuilding an index in SQL Data Warehouse is an offline operation. For more information about rebuilding indexes, see the ALTER INDEX REBUILD section in Columnstore Indexes Defragmentation, and ALTER INDEX.

Step 3: Verify clustered columnstore segment quality has improved

Rerun the query which identified table with poor segment quality and verify segment quality has improved. If segment quality did not improve, it could be that the rows in your table are extra wide. Consider using a higher resource class or DWU when rebuilding your indexes.

Rebuilding indexes with CTAS and partition switching

This example uses CTAS and partition switching to rebuild a table partition.

```
-- Step 1: Select the partition of data and write it out to a new table using CTAS
CREATE TABLE [dbo].[FactInternetSales_20000101_20010101]
    WITH    (   DISTRIBUTION = HASH([ProductKey])
            ,   CLUSTERED COLUMNSTORE INDEX
            ,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES
                                (20000101,20010101
                                )
                            )
            )
AS
SELECT  *
FROM    [dbo].[FactInternetSales]
WHERE   [OrderDateKey] >= 20000101
AND     [OrderDateKey] <  20010101
;

-- Step 2: Create a SWITCH out table
CREATE TABLE dbo.FactInternetSales_20000101
    WITH    (   DISTRIBUTION = HASH(ProductKey)
            ,   CLUSTERED COLUMNSTORE INDEX
            ,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES
                                (20000101
                                )
                            )
            )
AS
SELECT *
FROM    [dbo].[FactInternetSales]
WHERE   1=2 -- Note this table will be empty

-- Step 3: Switch OUT the data
ALTER TABLE [dbo].[FactInternetSales] SWITCH PARTITION 2 TO  [dbo].[FactInternetSales_20000101] PARTITION 2;

-- Step 4: Switch IN the rebuilt data
ALTER TABLE [dbo].[FactInternetSales_20000101_20010101] SWITCH PARTITION 2 TO  [dbo].[FactInternetSales]
PARTITION 2;
```

For more details about re-creating partitions using `CTAS`, see the Partition article.

Next steps

To learn more, see the articles on Table Overview, Table Data Types, Distributing a Table, Partitioning a Table, Maintaining Table Statistics and Temporary Tables. To learn more about best practices, see SQL Data Warehouse Best Practices.

Create surrogate keys by using IDENTITY

12/6/2017 • 6 min to read • Edit Online

Many data modelers like to create surrogate keys on their tables when they design data warehouse models. You can use the IDENTITY property to achieve this goal simply and effectively without affecting load performance.

Get started with IDENTITY

You can define a table as having the IDENTITY property when you first create the table by using syntax that is similar to the following statement:

```
CREATE TABLE dbo.T1
(   C1 INT IDENTITY(1,1) NOT NULL
,   C2 INT NULL
)
WITH
(   DISTRIBUTION = HASH(C2)
,   CLUSTERED COLUMNSTORE INDEX
)
;
```

You can then use `INSERT..SELECT` to populate the table.

Behavior

The IDENTITY property is designed to scale out across all the distributions in the data warehouse without affecting load performance. Therefore, the implementation of IDENTITY is oriented toward achieving these goals. This section highlights the nuances of the implementation to help you understand them more fully.

Allocation of values

The IDENTITY property doesn't guarantee the order in which the surrogate values are allocated, which reflects the behavior of SQL Server and Azure SQL Database. However, in Azure SQL Data Warehouse, the absence of a guarantee is more pronounced.

The following example is an illustration:

```
CREATE TABLE dbo.T1
(   C1 INT IDENTITY(1,1)    NOT NULL
,   C2 VARCHAR(30)              NULL
)
WITH
(   DISTRIBUTION = HASH(C2)
,   CLUSTERED COLUMNSTORE INDEX
)
;

INSERT INTO dbo.T1
VALUES (NULL);

INSERT INTO dbo.T1
VALUES (NULL);

SELECT *
FROM dbo.T1;

DBCC PDW_SHOWSPACEUSED('dbo.T1');
```

In the preceding example, two rows landed in distribution 1. The first row has the surrogate value of 1 in column `C1`, and the second row has the surrogate value of 61. Both of these values were generated by the IDENTITY property. However, the allocation of the values is not contiguous. This behavior is by design.

Skewed data

The range of values for the data type are spread evenly across the distributions. If a distributed table suffers from skewed data, then the range of values available to the datatype can be exhausted prematurely. For example, if all the data ends up in a single distribution, then effectively the table has access to only one-sixtieth of the values of the data type. For this reason, the IDENTITY property is limited to `INT` and `BIGINT` data types only.

SELECT..INTO

When an existing IDENTITY column is selected into a new table, the new column inherits the IDENTITY property, unless one of the following conditions is true:

- The SELECT statement contains a join.
- Multiple SELECT statements are joined by using UNION.
- The IDENTITY column is listed more than one time in the SELECT list.
- The IDENTITY column is part of an expression.

If any one of these conditions is true, the column is created NOT NULL instead of inheriting the IDENTITY property.

CREATE TABLE AS SELECT

CREATE TABLE AS SELECT (CTAS) follows the same SQL Server behavior that's documented for SELECT..INTO. However, you can't specify an IDENTITY property in the column definition of the `CREATE TABLE` part of the statement. You also can't use the IDENTITY function in the `SELECT` part of the CTAS. To populate a table, you need to use `CREATE TABLE` to define the table followed by `INSERT..SELECT` to populate it.

Explicitly insert values into an IDENTITY column

SQL Data Warehouse supports `SET IDENTITY_INSERT <your table> ON|OFF` syntax. You can use this syntax to explicitly insert values into the IDENTITY column.

Many data modelers like to use predefined negative values for certain rows in their dimensions. An example is the -1 or "unknown member" row.

The next script shows how to explicitly add this row by using SET IDENTITY_INSERT:

```
SET IDENTITY_INSERT dbo.T1 ON;

INSERT INTO dbo.T1
(   C1
,   C2
)
VALUES (-1,'UNKNOWN')
;

SET IDENTITY_INSERT dbo.T1 OFF;

SELECT  *
FROM    dbo.T1
;
```

Load data into a table with IDENTITY

The presence of the IDENTITY property has some implications to your data-loading code. This section highlights some basic patterns for loading data into tables by using IDENTITY.

Load data with PolyBase

To load data into a table and generate a surrogate key by using IDENTITY, create the table and then use INSERT..SELECT or INSERT..VALUES to perform the load.

The following example highlights the basic pattern:

```
--CREATE TABLE with IDENTITY
CREATE TABLE dbo.T1
(   C1 INT IDENTITY(1,1)
,   C2 VARCHAR(30)
)
WITH
(   DISTRIBUTION = HASH(C2)
,   CLUSTERED COLUMNSTORE INDEX
)
;

--Use INSERT..SELECT to populate the table from an external table
INSERT INTO dbo.T1
(C2)
SELECT  C2
FROM    ext.T1
;

SELECT  *
FROM    dbo.T1
;

DBCC PDW_SHOWSPACEUSED('dbo.T1');
```

> **NOTE**
>
> It's not possible to use `CREATE TABLE AS SELECT` currently when loading data into a table with an IDENTITY column.

For more information on loading data by using the bulk copy program (BCP) tool, see the following articles:

- Load with PolyBase
- PolyBase best practices

controls the behavior of BCP when loading data into a table with an IDENTITY column.

When -E is specified, the values held in the input file for the column with IDENTITY are retained. If -E is *not* specified, then the values in this column are ignored. If the identity column is not included, then the data is loaded as normal. The values are generated according to the increment and seed policy of the property.

For more information on loading data by using BCP, see the following articles:

- Load with BCP
- BCP in MSDN

Catalog views

SQL Data Warehouse supports the `sys.identity_columns` catalog view. This view can be used to identify a column that has the IDENTITY property.

To help you better understand the database schema, this example shows how to integrate `sys.identity_columns` with other system catalog views:

```
SELECT  sm.name
,       tb.name
,       co.name
,       CASE WHEN ic.column_id IS NOT NULL
             THEN 1
        ELSE 0
        END AS is_identity
FROM        sys.schemas AS sm
JOIN        sys.tables  AS tb           ON  sm.schema_id = tb.schema_id
JOIN        sys.columns AS co           ON  tb.object_id = co.object_id
LEFT JOIN   sys.identity_columns AS ic  ON  co.object_id = ic.object_id
                                        AND co.column_id = ic.column_id
WHERE   sm.name = 'dbo'
AND     tb.name = 'T1'
;
```

Limitations

The IDENTITY property can't be used in the following scenarios:

- Where the column data type is not INT or BIGINT
- Where the column is also the distribution key
- Where the table is an external table

The following related functions are not supported in SQL Data Warehouse:

- IDENTITY()
- @@IDENTITY
- SCOPE_IDENTITY
- IDENT_CURRENT
- IDENT_INCR
- IDENT_SEED
- DBCC CHECK_IDENT()

Tasks

> **NOTE**
> Column C1 is the IDENTITY in all the following tasks.

Find the highest allocated value for a table

Use the `MAX()` function to determine the highest value allocated for a distributed table:

```
SELECT  MAX(C1)
FROM    dbo.T1
```

Find the seed and increment for the IDENTITY property

You can use the catalog views to discover the identity increment and seed configuration values for a table by using the following query:

```
SELECT  sm.name
,       tb.name
,       co.name
,       ic.seed_value
,       ic.increment_value
FROM        sys.schemas AS sm
JOIN        sys.tables  AS tb           ON  sm.schema_id = tb.schema_id
JOIN        sys.columns AS co           ON  tb.object_id = co.object_id
JOIN        sys.identity_columns AS ic  ON  co.object_id = ic.object_id
                                        AND co.column_id = ic.column_id
WHERE   sm.name = 'dbo'
AND     tb.name = 'T1'
;
```

Next steps

- To learn more about developing tables, see Table overview, Table data types, Distribute a table, Index a table, Partition a table, and Temporary tables.
- For more information about best practices, see SQL Data Warehouse best practices.

Partitioning tables in SQL Data Warehouse

12/6/2017 • 11 min to read • Edit Online

Partitioning is supported on all SQL Data Warehouse table types; including clustered columnstore, clustered index, and heap. Partitioning is also supported on all distribution types, including both hash or round robin distributed. Partitioning enables you to divide your data into smaller groups of data and in most cases, partitioning is done on a date column.

Benefits of partitioning

Partitioning can benefit data maintenance and query performance. Whether it benefits both or just one is dependent on how data is loaded and whether the same column can be used for both purposes, since partitioning can only be done on one column.

Benefits to loads

The primary benefit of partitioning in SQL Data Warehouse is to improve the efficiency and performance of loading data by use of partition deletion, switching and merging. In most cases data is partitioned on a date column that is closely tied to the order in which the data is loaded into the database. One of the greatest benefits of using partitions to maintain data it the avoidance of transaction logging. While simply inserting, updating, or deleting data can be the most straightforward approach, with a little thought and effort, using partitioning during your load process can substantially improve performance.

Partition switching can be used to quickly remove or replace a section of a table. For example, a sales fact table might contain just data for the past 36 months. At the end of every month, the oldest month of sales data is deleted from the table. This data could be deleted by using a delete statement to delete the data for the oldest month. However, deleting a large amount of data row-by-row with a delete statement can take too much time, as well as create the risk of large transactions that take a long time to rollback if something goes wrong. A more optimal approach is to drop the oldest partition of data. Where deleting the individual rows could take hours, deleting an entire partition could take seconds.

Benefits to queries

Partitioning can also be used to improve query performance. A query that applies a filter to partitioned data can limit the scan to only the qualifying partitions. This method of filtering can avoid a full table scan and only scan a smaller subset of data. With the introduction of clustered columnstore indexes, the predicate elimination performance benefits are less beneficial, but in some cases there can be a benefit to queries. For example, if the sales fact table is partitioned into 36 months using the sales date field, then queries that filter on the sale date can skip searching in partitions that don't match the filter.

Partition sizing guidance

While partitioning can be used to improve performance some scenarios, creating a table with **too many** partitions can hurt performance under some circumstances. These concerns are especially true for clustered columnstore tables. For partitioning to be helpful, it is important to understand when to use partitioning and the number of partitions to create. There is no hard fast rule as to how many partitions are too many, it depends on your data and how many partitions you are loading to simultaneously. A successful partitioning scheme usually has tens to hundreds of partitions, not thousands.

When creating partitions on **clustered columnstore** tables, it is important to consider how many rows belong to each partition. For optimal compression and performance of clustered columnstore tables, a minimum of 1 million rows per distribution and partition is needed. Before partitions are created, SQL Data Warehouse

distributions created behind the scenes. Using this example, if the sales fact table contained 36 monthly partitions, and given that SQL Data Warehouse has 60 distributions, then the sales fact table should contain 60 million rows per month, or 2.1 billion rows when all months are populated. If a table contains significantly fewer than the recommended minimum number of rows per partition, consider using fewer partitions in order to increase the number of rows per partition. Also see the Indexing article, which includes queries that can be run on SQL Data Warehouse to assess the quality of cluster columnstore indexes.

Syntax difference from SQL Server

SQL Data Warehouse introduces a simplified way to define partitions, which is slightly different from SQL Server. Partitioning functions and schemes are not used in SQL Data Warehouse as they are in SQL Server. Instead, all you need to do is identify partitioned column and the boundary points. While the syntax of partitioning may be slightly different from SQL Server, the basic concepts are the same. SQL Server and SQL Data Warehouse support one partition column per table, which can be ranged partition. To learn more about partitioning, see Partitioned Tables and Indexes.

The following example of a SQL Data Warehouse partitioned CREATE TABLE statement, partitions the FactInternetSales table on the OrderDateKey column:

```
CREATE TABLE [dbo].[FactInternetSales]
(
    [ProductKey]            int          NOT NULL
,   [OrderDateKey]          int          NOT NULL
,   [CustomerKey]           int          NOT NULL
,   [PromotionKey]          int          NOT NULL
,   [SalesOrderNumber]      nvarchar(20) NOT NULL
,   [OrderQuantity]         smallint     NOT NULL
,   [UnitPrice]             money        NOT NULL
,   [SalesAmount]           money        NOT NULL
)
WITH
(   CLUSTERED COLUMNSTORE INDEX
,   DISTRIBUTION = HASH([ProductKey])
,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES
                    (20000101,20010101,20020101
                    ,20030101,20040101,20050101
                    )
                )
)
;
```

Migrating partitioning from SQL Server

To migrate SQL Server partition definitions to SQL Data Warehouse simply:

- Eliminate the SQL Server partition scheme.
- Add the partition function definition to your CREATE TABLE.

If you are migrating a partitioned table from a SQL Server instance, the following SQL can help you to figure out the number of rows that in each partition. Keep in mind that if the same partitioning granularity is used on SQL Data Warehouse, the number of rows per partition decreases by a factor of 60.

```
-- Partition information for a SQL Server Database
SELECT      s.[name]                        AS      [schema_name]
,           t.[name]                        AS      [table_name]
,           i.[name]                        AS      [index_name]
,           p.[partition_number]            AS      [partition_number]
,           SUM(a.[used_pages]*8.0)         AS      [partition_size_kb]
,           SUM(a.[used_pages]*8.0)/1024    AS      [partition_size_mb]
,           SUM(a.[used_pages]*8.0)/1048576 AS      [partition_size_gb]
,           p.[rows]                        AS      [partition_row_count]
,           rv.[value]                      AS      [partition_boundary_value]
,           p.[data_compression_desc]       AS      [partition_compression_desc]
FROM        sys.schemas s
JOIN        sys.tables t                    ON      t.[schema_id]         = s.[schema_id]
JOIN        sys.partitions p                ON      p.[object_id]         = t.[object_id]
JOIN        sys.allocation_units a          ON      a.[container_id]      = p.[partition_id]
JOIN        sys.indexes i                   ON      i.[object_id]         = p.[object_id]
                                            AND     i.[index_id]          = p.[index_id]
JOIN        sys.data_spaces ds              ON      ds.[data_space_id]    = i.[data_space_id]
LEFT JOIN   sys.partition_schemes ps        ON      ps.[data_space_id]    = ds.[data_space_id]
LEFT JOIN   sys.partition_functions pf      ON      pf.[function_id]      = ps.[function_id]
LEFT JOIN   sys.partition_range_values rv   ON      rv.[function_id]      = pf.[function_id]
                                            AND     rv.[boundary_id]      = p.[partition_number]
WHERE       p.[index_id] <=1
GROUP BY    s.[name]
,           t.[name]
,           i.[name]
,           p.[partition_number]
,           p.[rows]
,           rv.[value]
,           p.[data_compression_desc]
;
```

Workload management

One final piece consideration to factor in to the table partition decision is workload management. Workload management in SQL Data Warehouse is primarily the management of memory and concurrency. In SQL Data Warehouse, the maximum memory allocated to each distribution during query execution is governed by resource classes. Ideally your partitions are sized in consideration of other factors like the memory needs of building clustered columnstore indexes. Clustered columnstore indexes benefit greatly when they are allocated more memory. Therefore, you want to ensure that a partition index rebuild is not starved of memory. Increasing the amount of memory available to your query can be achieved by switching from the default role, smallrc, to one of the other roles such as largerc.

Information on the allocation of memory per distribution is available by querying the resource governor dynamic management views. In reality, your memory grant is less than the following figures. However, this provides a level of guidance that you can use when sizing your partitions for data management operations. Try to avoid sizing your partitions beyond the memory grant provided by the extra large resource class. If your partitions grow beyond this figure you run the risk of memory pressure which in turn leads to less optimal compression.

```
SELECT  rp.[name]                                   AS [pool_name]
,       rp.[max_memory_kb]                           AS [max_memory_kb]
,       rp.[max_memory_kb]/1024                      AS [max_memory_mb]
,       rp.[max_memory_kb]/1048576                   AS [mex_memory_gb]
,       rp.[max_memory_percent]                      AS [max_memory_percent]
,       wg.[name]                                   AS [group_name]
,       wg.[importance]                              AS [group_importance]
,       wg.[request_max_memory_grant_percent]       AS [request_max_memory_grant_percent]
FROM    sys.dm_pdw_nodes_resource_governor_workload_groups    wg
JOIN    sys.dm_pdw_nodes_resource_governor_resource_pools    rp ON wg.[pool_id] = rp.[pool_id]
WHERE   wg.[name] like 'SloDWGroup%'
AND     rp.[name]    = 'SloDWPool'
;
```

Partition switching

SQL Data Warehouse supports partition splitting, merging, and switching. Each of these functions is excuted using the ALTER TABLE statement.

To switch partitions between two tables you must ensure that the partitions align on their respective boundaries and that the table definitions match. As check constraints are not available to enforce the range of values in a table the source table must contain the same partition boundaries as the target table. If this is not the case, then the partition switch will fail as the partition metadata will not be synchronized.

How to split a partition that contains data

The most efficient method to split a partition that already contains data is to use a `CTAS` statement. If the partitioned table is a clustered columnstore then the table partition must be empty before it can be split.

Below is a sample partitioned columnstore table containing one row in each partition:

```
CREATE TABLE [dbo].[FactInternetSales]
(
        [ProductKey]            int          NOT NULL
    ,   [OrderDateKey]          int          NOT NULL
    ,   [CustomerKey]           int          NOT NULL
    ,   [PromotionKey]          int          NOT NULL
    ,   [SalesOrderNumber]      nvarchar(20) NOT NULL
    ,   [OrderQuantity]         smallint     NOT NULL
    ,   [UnitPrice]             money        NOT NULL
    ,   [SalesAmount]           money        NOT NULL
)
WITH
(   CLUSTERED COLUMNSTORE INDEX
,   DISTRIBUTION = HASH([ProductKey])
,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES
                    (20000101
                    )
                )
)
;

INSERT INTO dbo.FactInternetSales
VALUES (1,19990101,1,1,1,1,1,1);
INSERT INTO dbo.FactInternetSales
VALUES (1,20000101,1,1,1,1,1,1);


CREATE STATISTICS Stat_dbo_FactInternetSales_OrderDateKey ON dbo.FactInternetSales(OrderDateKey);
```

NOTE

By Creating the statistic object, we ensure that table metadata is more accurate. If we omit creating statistics, then SQL Data Warehouse will use default values. For details on statistics please review statistics.

We can then query for the row count using the `sys.partitions` catalog view:

```
SELECT  QUOTENAME(s.[name])+'.'+QUOTENAME(t.[name]) as Table_name
,       i.[name] as Index_name
,       p.partition_number as Partition_nmbr
,       p.[rows] as Row_count
,       p.[data_compression_desc] as Data_Compression_desc
FROM    sys.partitions p
JOIN    sys.tables     t    ON    p.[object_id]   = t.[object_id]
JOIN    sys.schemas    s    ON    t.[schema_id]   = s.[schema_id]
JOIN    sys.indexes    i    ON    p.[object_id]   = i.[object_Id]
                            AND   p.[index_Id]    = i.[index_Id]
WHERE t.[name] = 'FactInternetSales'
;
```

If we try to split this table, we will get an error:

```
ALTER TABLE FactInternetSales SPLIT RANGE (20010101);
```

Msg 35346, Level 15, State 1, Line 44 SPLIT clause of ALTER PARTITION statement failed because the partition is not empty. Only empty partitions can be split in when a columnstore index exists on the table. Consider disabling the columnstore index before issuing the ALTER PARTITION statement, then rebuilding the columnstore index after ALTER PARTITION is complete.

However, we can use `CTAS` to create a new table to hold our data.

```
CREATE TABLE dbo.FactInternetSales_20000101
    WITH    (   DISTRIBUTION = HASH(ProductKey)
            ,   CLUSTERED COLUMNSTORE INDEX
            ,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES
                                (20000101
                                )
                            )
            )
AS
SELECT *
FROM    FactInternetSales
WHERE   1=2
;
```

As the partition boundaries are aligned a switch is permitted. This will leave the source table with an empty partition that we can subsequently split.

```
ALTER TABLE FactInternetSales SWITCH PARTITION 2 TO  FactInternetSales_20000101 PARTITION 2;

ALTER TABLE FactInternetSales SPLIT RANGE (20010101);
```

All that is left to do is to align our data to the new partition boundaries using `CTAS` and switch our data back in to the main table

```
CREATE TABLE [dbo].[FactInternetSales_20000101_20010101]
    WITH    (   DISTRIBUTION = HASH([ProductKey])
            ,   CLUSTERED COLUMNSTORE INDEX
            ,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES
                                (20000101,20010101
                                )
                            )
            )
AS
SELECT  *
FROM    [dbo].[FactInternetSales_20000101]
WHERE   [OrderDateKey] >= 20000101
AND     [OrderDateKey] <  20010101
;

ALTER TABLE dbo.FactInternetSales_20000101_20010101 SWITCH PARTITION 2 TO dbo.FactInternetSales PARTITION
2;
```

Once you have completed the movement of the data it is a good idea to refresh the statistics on the target table to ensure they accurately reflect the new distribution of the data in their respective partitions:

```
UPDATE STATISTICS [dbo].[FactInternetSales];
```

Table partitioning source control

To avoid your table definition from **rusting** in your source control system you may want to consider the following approach:

1. Create the table as a partitioned table but with no partition values

```
CREATE TABLE [dbo].[FactInternetSales]
(
    [ProductKey]            int          NOT NULL
,   [OrderDateKey]          int          NOT NULL
,   [CustomerKey]           int          NOT NULL
,   [PromotionKey]          int          NOT NULL
,   [SalesOrderNumber]      nvarchar(20) NOT NULL
,   [OrderQuantity]         smallint     NOT NULL
,   [UnitPrice]             money        NOT NULL
,   [SalesAmount]           money        NOT NULL
)
WITH
(   CLUSTERED COLUMNSTORE INDEX
,   DISTRIBUTION = HASH([ProductKey])
,   PARTITION   (   [OrderDateKey] RANGE RIGHT FOR VALUES
                    ()
                )
)
;
```

1. `SPLIT` the table as part of the deployment process:

```
-- Create a table containing the partition boundaries

CREATE TABLE #partitions
WITH
(
    LOCATION = USER_DB
,   DISTRIBUTION = HASH(ptn_no)
)
AS
SELECT  ptn_no
,       ROW_NUMBER() OVER (ORDER BY (ptn_no)) as seq_no
FROM    (
        SELECT CAST(20000101 AS INT) ptn_no
        UNION ALL
        SELECT CAST(20010101 AS INT)
        UNION ALL
        SELECT CAST(20020101 AS INT)
        UNION ALL
        SELECT CAST(20030101 AS INT)
        UNION ALL
        SELECT CAST(20040101 AS INT)
        ) a
;

-- Iterate over the partition boundaries and split the table

DECLARE @c INT = (SELECT COUNT(*) FROM #partitions)
,       @i INT = 1                              --iterator for while loop
,       @q NVARCHAR(4000)                       --query
,       @p NVARCHAR(20)     = N''               --partition_number
,       @s NVARCHAR(128)    = N'dbo'            --schema
,       @t NVARCHAR(128)    = N'FactInternetSales' --table
;

WHILE @i <= @c
BEGIN
    SET @p = (SELECT ptn_no FROM #partitions WHERE seq_no = @i);
    SET @q = (SELECT N'ALTER TABLE '+@s+N'.'+@t+N' SPLIT RANGE ('+@p+N');');

    -- PRINT @q;
    EXECUTE sp_executesql @q;

    SET @i+=1;
END

-- Code clean-up

DROP TABLE #partitions;
```

With this approach the code in source control remains static and the partitioning boundary values are allowed to be dynamic; evolving with the warehouse over time.

Next steps

To learn more, see the articles on Table Overview, Table Data Types, Distributing a Table, Indexing a Table, Maintaining Table Statistics and Temporary Tables. For more about best practices, see SQL Data Warehouse Best Practices.

Design guidance for using replicated tables in Azure SQL Data Warehouse

12/7/2017 • 7 min to read • Edit Online

This article gives recommendations for designing replicated tables in your SQL Data Warehouse schema. Use these recommendations to improve query performance by reducing data movement and query complexity.

> **NOTE**
> The replicated table feature is currently in public preview. Some behaviors are subject to change.

Prerequisites

This article assumes you are familiar with data distribution and data movement concepts in SQL Data Warehouse. For more information, see the architecture article.

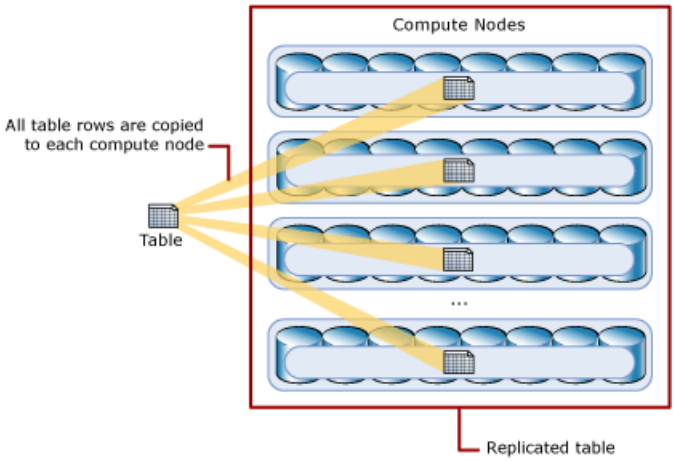
As part of table design, understand as much as possible about your data and how the data is queried. For example, consider these questions:

- How large is the table?
- How often is the table refreshed?
- Do I have fact and dimension tables in a data warehouse?

What is a replicated table?

A replicated table has a full copy of the table accessible on each Compute node. Replicating a table removes the need to transfer data among Compute nodes before a join or aggregation. Since the table has multiple copies, replicated tables work best when the table size is less than 2 GB compressed.

The following diagram shows a replicated table that is accessible on each Compute node. In SQL Data Warehouse, the replicated table is fully copied to a distribution database on each Compute node.



Replicated tables work well for small dimension tables in a star schema. Dimension tables are usually of a size that makes it feasible to store and maintain multiple copies. Dimensions store descriptive data that changes slowly, such as customer name and address, and product details. The slowly changing nature of the data leads to fewer rebuilds of the replicated table.

- The table size on disk is less than 2 GB, regardless of the number of rows. To find the size of a table, you can use the DBCC PDW_SHOWSPACEUSED command: `DBCC PDW_SHOWSPACEUSED('ReplTableCandidate')`.
- The table is used in joins that would otherwise require data movement. For example, a join on hash-distributed tables requires data movement when the joining columns are not the same distribution column. If one of the hash-distributed tables is small, consider a replicated table. A join on a round-robin table requires data movement. We recommend using replicated tables instead of round-robin tables in most cases.

Consider converting an existing distributed table to a replicated table when:

- Query plans use data movement operations that broadcast the data to all the Compute nodes. The BroadcastMoveOperation is expensive and slows query performance. To view data movement operations in query plans, use sys.dm_pdw_request_steps.

Replicated tables may not yield the best query performance when:

- The table has frequent insert, update, and delete operations. These data manipulation language (DML) operations require a rebuild of the replicated table. Rebuilding frequently can cause slower performance.
- The data warehouse is scaled frequently. Scaling a data warehouse changes the number of Compute nodes, which incurs a rebuild.
- The table has a large number of columns, but data operations typically access only a small number of columns. In this scenario, instead of replicating the entire table, it might be more effective to hash distribute the table, and then create an index on the frequently accessed columns. When a query requires data movement, SQL Data Warehouse only moves data in the requested columns.

Use replicated tables with simple query predicates

Before you choose to distribute or replicate a table, think about the types of queries you plan to run against the table. Whenever possible,

- Use replicated tables for queries with simple query predicates, such as equality or inequality.
- Use distributed tables for queries with complex query predicates, such as LIKE or NOT LIKE.

CPU-intensive queries perform best when the work is distributed across all of the Compute nodes. For example, queries that run computations on each row of a table perform better on distributed tables than replicated tables. Since a replicated table is stored in full on each Compute node, a CPU-intensive query against a replicated table runs against the entire table on every Compute node. The extra computation can slow query performance.

For example, this query has a complex predicate. It runs faster when supplier is a distributed table instead of a replicated table. In this example, supplier can be hash-distributed or round-robin distributed.

```
SELECT EnglishProductName
FROM DimProduct
WHERE EnglishDescription LIKE '%frame%comfortable%'
```

Convert existing round-robin tables to replicated tables

If you already have round-robin tables, we recommend converting them to replicated tables if they meet with criteria outlined in this article. Replicated tables improve performance over round-robin tables because they eliminate the need for data movement. A round-robin table always requires data movement for joins.

This example uses CTAS to change the DimSalesTerritory table to a replicated table. This example works regardless of whether DimSalesTerritory is hash-distributed or round-robin.

```
CREATE TABLE [dbo].[DimSalesTerritory_REPLICATE]
WITH
  (
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = REPLICATE
  )
AS SELECT * FROM [dbo].[DimSalesTerritory]
OPTION  (LABEL  = 'CTAS : DimSalesTerritory_REPLICATE')

--Create statistics on new table
CREATE STATISTICS [SalesTerritoryKey] ON [DimSalesTerritory_REPLICATE] ([SalesTerritoryKey]);
CREATE STATISTICS [SalesTerritoryAlternateKey] ON [DimSalesTerritory_REPLICATE] ([SalesTerritoryAlternateKey]);
CREATE STATISTICS [SalesTerritoryRegion] ON [DimSalesTerritory_REPLICATE] ([SalesTerritoryRegion]);
CREATE STATISTICS [SalesTerritoryCountry] ON [DimSalesTerritory_REPLICATE] ([SalesTerritoryCountry]);
CREATE STATISTICS [SalesTerritoryGroup] ON [DimSalesTerritory_REPLICATE] ([SalesTerritoryGroup]);

-- Switch table names
RENAME OBJECT [dbo].[DimSalesTerritory] to [DimSalesTerritory_old];
RENAME OBJECT [dbo].[DimSalesTerritory_REPLICATE] TO [DimSalesTerritory];

DROP TABLE [dbo].[DimSalesTerritory_old];
```

Query performance example for round-robin versus replicated

A replicated table does not require any data movement for joins because the entire table is already present on each Compute node. If the dimension tables are round-robin distributed, a join copies the dimension table in full to each Compute node. To move the data, the query plan contains an operation called BroadcastMoveOperation. This type of data movement operation slows query performance and is eliminated by using replicated tables. To view query plan steps, use the sys.dm_pdw_request_steps system catalog view.

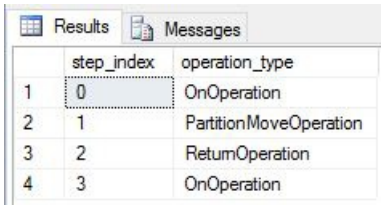
For example, in following query against the AdventureWorks schema, the `FactInternetSales` table is hash-distributed. The `DimDate` and `DimSalesTerritory` tables are smaller dimension tables. This query returns the total sales in North America for fiscal year 2004:

```
SELECT [TotalSalesAmount] = SUM(SalesAmount)
FROM dbo.FactInternetSales s
INNER JOIN dbo.DimDate d
  ON d.DateKey = s.OrderDateKey
INNER JOIN dbo.DimSalesTerritory t
  ON t.SalesTerritoryKey = s.SalesTerritoryKey
WHERE d.FiscalYear = 2004
  AND t.SalesTerritoryGroup = 'North America'
```

We re-created `DimDate` and `DimSalesTerritory` as round-robin tables. As a result, the query showed the following query plan, which has multiple broadcast move operations:

Results | Messages

	step_index	operation_type
1	0	RandomIDOperation
2	1	OnOperation
3	2	BroadcastMoveOperation
4	3	RandomIDOperation
5	4	OnOperation
6	5	BroadcastMoveOperation
7	6	OnOperation
8	7	PartitionMoveOperation
9	8	OnOperation
10	9	OnOperation
11	10	ReturnOperation
12	11	OnOperation



Results | Messages

	step_index	operation_type
1	0	OnOperation
2	1	PartitionMoveOperation
3	2	ReturnOperation
4	3	OnOperation

Performance considerations for modifying replicated tables

SQL Data Warehouse implements a replicated table by maintaining a master version of the table. It copies the master version to one distribution database on each Compute node. When there is a change, SQL Data Warehouse first updates the master table. Then it requires a rebuild of the tables on each Compute node. A rebuild of a replicated table includes copying the table to each Compute node and then rebuilding the indexes.

Rebuilds are required after:

- Data is loaded or modified
- The data warehouse is scaled to a different service level
- Table definition is updated

Rebuilds are not required after:

- Pause operation
- Resume operation

The rebuild does not happen immediately after data is modified. Instead, the rebuild is triggered the first time a query selects from the table. Within the initial select statement from the table are steps to rebuild the replicated table. Because the rebuild is done within the query, the impact to the initial select statement could be significant depending on the size of the table. If multiple replicated tables are involved that need a rebuild, each copy is rebuilt serially as steps within the statement. To maintain data consistency during the rebuild of the replicated table an exclusive lock is taken on the table. The lock prevents all access to the table for the duration of the rebuild.

Use indexes conservatively

Standard indexing practices apply to replicated tables. SQL Data Warehouse rebuilds each replicated table index as part of the rebuild. Only use indexes when the performance gain outweighs the cost of rebuilding the indexes.

Batch data loads

When loading data into replicated tables, try to minimize rebuilds by batching loads together. Perform all the batched loads before running select statements.

For example, this load pattern loads data from four sources and invokes four rebuilds.

- Load from source 1.
- Select statement triggers rebuild 1.
- Load from source 2.
- Select statement triggers rebuild 2.
- Load from source 3.
- Select statement triggers rebuild 3.
- Load from source 4.
- Select statement triggers rebuild 4.

For example, this load pattern loads data from four sources, but only invokes one rebuild.

- Load from source 1.

- Load from source 3.
- Load from source 4.
- Select statement triggers rebuild.

Rebuild a replicated table after a batch load

To ensure consistent query execution times, we recommend forcing a refresh of the replicated tables after a batch load. Otherwise, the first query must wait for the tables to refresh, which includes rebuilding the indexes. Depending on the size and number of replicated tables affected, the performance impact can be significant.

This query uses the sys.pdw_replicated_table_cache_state DMV to list the replicated tables that have been modified, but not rebuilt.

```
SELECT [ReplicatedTable] = t.[name]
  FROM sys.tables t
  JOIN sys.pdw_replicated_table_cache_state c
    ON c.object_id = t.object_id
  JOIN sys.pdw_table_distribution_properties p
    ON p.object_id = t.object_id
  WHERE c.[state] = 'NotReady'
    AND p.[distribution_policy_desc] = 'REPLICATE'
```

To force a rebuild, run the following statement on each table in the preceding output.

```
SELECT TOP 1 * FROM [ReplicatedTable]
```

Next steps

To create a replicated table, use one of these statements:

- CREATE TABLE (Azure SQL Data Warehouse)
- CREATE TABLE AS SELECT (Azure SQL Data Warehouse)

For an overview of distributed tables, see distributed tables.

Managing statistics on tables in SQL Data Warehouse

12/21/2017 • 13 min to read • Edit Online

The more SQL Data Warehouse knows about your data, the faster it can execute queries against your data. The way that you tell SQL Data Warehouse about your data, is by collecting statistics about your data. Having statistics on your data is one of the most important things you can do to optimize your queries. This is because the SQL Data Warehouse query optimizer is a cost-based optimizer. It compares the cost of various query plans and then chooses the plan with the lowest cost, which should also be the plan that will execute the fastest. For example, if the optimizer estimates that the date you are filtering in your query will return 1 row, it may choose a very different plan than if it estimates that they date you have selected will return 1 million rows.

The process of creating and updating statistics is currently a manual process, but is very simple to do. Soon youw will be able to automatically create and update statistics on single columns and indexes. By using the information below, you can greatly automate the management of the statistics on your data.

Getting started with statistics

Creating sampled statistics on every column is an easy way to get started with statistics. Out-of-date statistics will lead to sub-optimal query performance. However it can consume memory to update statistics on all columns as your data grow.

Following are some of recommendations for different scenarios:

SCENARIOS	RECOMMENDATION
Get started	Update all columns after migrating to SQL DW
Most important column for stats	Hash distribution key
Second most important column for stats	Partition Key
Other important columns for stats	Date, Frequent JOINs, GROUP BY, HAVING and WHERE
Frequency of stats updates	Conservative: Daily After loading or transforming your data
Sampling	Below 1 B rows, use default sampling (20%) With more than 1 B rows tables, statistics on a 2% range is good

Updating statistics

One best practice is to update statistics on date columns each day as new dates are added. Each time new rows are loaded into the data warehouse, new load dates or transaction dates are added. These change the data distribution and make the statistics out-of-date. Conversely, statistics on a country column in a customer table might never need to be updated, as the distribution of values doesn't generally change. Assuming the distribution is constant between customers, adding new rows to the table variation isn't going to change the data distribution. However, if your data warehouse only contains one country and you bring in data from a

statistics on the country column.

One of the first questions to ask when troubleshooting a query is, **"Are the statistics up-to-date?"**

This question is not one that can be answered by the age of the data. An up to date statistics object could be very old if there's been no material change to the underlying data. When the number of rows has changed substantially or there is a material change in the distribution of values for a given column *then* it's time to update statistics.

Since there is no DMV to determine if data within the table has changed since the last time statistics were updated, knowing the age of your statistics can provide you with part of the picture. You can use the following query to determine the last time your statistics where updated on each table.

> **NOTE**
> Remember if there is a material change in the distribution of values for a given column, you should update statistics regardless of the last time they were updated.

```
SELECT
    sm.[name] AS [schema_name],
    tb.[name] AS [table_name],
    co.[name] AS [stats_column_name],
    st.[name] AS [stats_name],
    STATS_DATE(st.[object_id],st.[stats_id]) AS [stats_last_updated_date]
FROM
    sys.objects ob
    JOIN sys.stats st
        ON  ob.[object_id] = st.[object_id]
    JOIN sys.stats_columns sc
        ON  st.[stats_id] = sc.[stats_id]
        AND st.[object_id] = sc.[object_id]
    JOIN sys.columns co
        ON  sc.[column_id] = co.[column_id]
        AND sc.[object_id] = co.[object_id]
    JOIN sys.types  ty
        ON  co.[user_type_id] = ty.[user_type_id]
    JOIN sys.tables tb
        ON  co.[object_id] = tb.[object_id]
    JOIN sys.schemas sm
        ON  tb.[schema_id] = sm.[schema_id]
WHERE
    st.[user_created] = 1;
```

Date columns in a data warehouse, for example, usually need frequent statistics updates. Each time new rows are loaded into the data warehouse, new load dates or transaction dates are added. These change the data distribution and make the statistics out-of-date. Conversely, statistics on a gender column on a customer table might never need to be updated. Assuming the distribution is constant between customers, adding new rows to the table variation isn't going to change the data distribution. However, if your data warehouse only contains one gender and a new requirement results in multiple genders then you definitely need to update statistics on the gender column.

For further explanation, see Statistics on MSDN.

Implementing statistics management

It is often a good idea to extend your data loading process to ensure that statistics are updated at the end of the load. The data load is when tables most frequently change their size and/or their distribution of values.

Some guiding principles are provided below for updating your statistics during the load process:

- Ensure that each loaded table has at least one statistics object updated. This updates the tables size (row count and page count) information as part of the stats update.
- Focus on columns participating in JOIN, GROUP BY, ORDER BY and DISTINCT clauses
- Consider updating "ascending key" columns such as transaction dates more frequently as these values will not be included in the statistics histogram.
- Consider updating static distribution columns less frequently.
- Remember each statistic object is updated in series. Simply implementing `UPDATE STATISTICS <TABLE_NAME>` may not be ideal - especially for wide tables with lots of statistics objects.

> **NOTE**
> For more details on [ascending key] please refer to the SQL Server 2014 cardinality estimation model whitepaper.

For further explanation, see Cardinality Estimation on MSDN.

Examples: Create statistics

These examples show how to use various options for creating statistics. The options that you use for each column depend on the characteristics of your data and how the column will be used in queries.

A. Create single-column statistics with default options

To create statistics on a column, simply provide a name for the statistics object and the name of the column.

This syntax uses all of the default options. By default, SQL Data Warehouse **samples 20 percent** of the table when it creates statistics.

```
CREATE STATISTICS [statistics_name] ON [schema_name].[table_name]([column_name]);
```

For example:

```
CREATE STATISTICS col1_stats ON dbo.table1 (col1);
```

B. Create single-column statistics by examining every row

The default sampling rate of 20 percent is sufficient for most situations. However, you can adjust the sampling rate.

To sample the full table, use this syntax:

```
CREATE STATISTICS [statistics_name] ON [schema_name].[table_name]([column_name]) WITH FULLSCAN;
```

For example:

```
CREATE STATISTICS col1_stats ON dbo.table1 (col1) WITH FULLSCAN;
```

C. Create single-column statistics by specifying the sample size

Alternatively, you can specify the sample size as a percent:

```
CREATE STATISTICS col1_stats ON dbo.table1 (col1) WITH SAMPLE = 50 PERCENT;
```

D. Create single-column statistics on only some of the rows

Another option, you can create statistics on a portion of the rows in your table. This is called a filtered statistic.

For example, you could use filtered statistics when you plan to query a specific partition of a large partitioned table. By creating statistics on only the partition values, the accuracy of the statistics will improve, and therefore improve query performance.

This example creates statistics on a range of values. The values could easily be defined to match the range of values in a partition.

```
CREATE STATISTICS stats_col1 ON table1(col1) WHERE col1 > '2000101' AND col1 < '20001231';
```

> **NOTE**
> For the query optimizer to consider using filtered statistics when it chooses the distributed query plan, the query must fit inside the definition of the statistics object. Using the previous example, the query's where clause needs to specify col1 values between 2000101 and 20001231.

E. Create single-column statistics with all the options

You can, of course, combine the options together. The example below creates a filtered statistics object with a custom sample size:

```
CREATE STATISTICS stats_col1 ON table1 (col1) WHERE col1 > '2000101' AND col1 < '20001231' WITH SAMPLE =
50 PERCENT;
```

For the full reference, see CREATE STATISTICS on MSDN.

F. Create multi-column statistics

To create a multi-column statistics, simply use the previous examples, but specify more columns.

> **NOTE**
> The histogram, which is used to estimate number of rows in the query result, is only available for the first column listed in the statistics object definition.

In this example, the histogram is on *product_category*. Cross-column statistics are calculated on *product_category* and *product_sub_category*:

```
CREATE STATISTICS stats_2cols ON table1 (product_category, product_sub_category) WHERE product_category >
'2000101' AND product_category < '20001231' WITH SAMPLE = 50 PERCENT;
```

Since there is a correlation between *product_category* and *product_sub_category*, a multi-column stat can be useful if these columns are accessed at the same time.

G. Create statistics on all the columns in a table

One way to create statistics is to issues CREATE STATISTICS commands after creating the table.

```
CREATE TABLE dbo.table1
(
   col1 int
,  col2 int
,  col3 int
)
WITH
  (
    CLUSTERED COLUMNSTORE INDEX
  )
;

CREATE STATISTICS stats_col1 on dbo.table1 (col1);
CREATE STATISTICS stats_col2 on dbo.table2 (col2);
CREATE STATISTICS stats_col3 on dbo.table3 (col3);
```

H. Use a stored procedure to create statistics on all columns in a database

SQL Data Warehouse does not have a system stored procedure equivalent to [sp_create_stats][] in SQL Server. This stored procedure creates a single column statistics object on every column of the database that doesn't already have statistics.

This will help you get started with your database design. Feel free to adapt it to your needs.

```
CREATE PROCEDURE    [dbo].[prc_sqldw_create_stats]
(   @create_type    tinyint -- 1 default 2 Fullscan 3 Sample
,   @sample_pct     tinyint
)
AS

IF @create_type NOT IN (1,2,3)
BEGIN
    THROW 151000,'Invalid value for @stats_type parameter. Valid range 1 (default), 2 (fullscan) or 3
(sample).',1;
END;

IF @sample_pct IS NULL
BEGIN;
    SET @sample_pct = 20;
END;

IF OBJECT_ID('tempdb..#stats_ddl') IS NOT NULL
BEGIN;
    DROP TABLE #stats_ddl;
END;

CREATE TABLE #stats_ddl
WITH    (   DISTRIBUTION    = HASH([seq_nmbr])
        ,   LOCATION        = USER_DB
        )
AS
WITH T
AS
(
SELECT      t.[name]                        AS [table_name]
,           s.[name]                        AS [table_schema_name]
,           c.[name]                        AS [column_name]
,           c.[column_id]                   AS [column_id]
,           t.[object_id]                   AS [object_id]
,           ROW_NUMBER()
            OVER(ORDER BY (SELECT NULL))    AS [seq_nmbr]
FROM        sys.[tables] t
JOIN        sys.[schemas] s         ON  t.[schema_id]       = s.[schema_id]
JOIN        sys.[columns] c         ON  t.[object_id]       = c.[object_id]
LEFT JOIN   sys.[stats_columns] l   ON  l.[object_id]       = c.[object_id]
```

```
LEFT JOIN    sys.[external_tables] e    ON    e.[object_id]        = t.[object_id]
WHERE       l.[object_id] IS NULL
AND            e.[object_id] IS NULL -- not an external table
)
SELECT  [table_schema_name]
,       [table_name]
,       [column_name]
,       [column_id]
,       [object_id]
,       [seq_nmbr]
,       CASE @create_type
        WHEN 1
        THEN    CAST('CREATE STATISTICS '+QUOTENAME('stat_'+table_schema_name+ '_' + table_name +
'_'+column_name)+' ON
'+QUOTENAME(table_schema_name)+'.'+QUOTENAME(table_name)+'('+QUOTENAME(column_name)+')' AS VARCHAR(8000))
        WHEN 2
        THEN    CAST('CREATE STATISTICS '+QUOTENAME('stat_'+table_schema_name+ '_' + table_name +
'_'+column_name)+' ON
'+QUOTENAME(table_schema_name)+'.'+QUOTENAME(table_name)+'('+QUOTENAME(column_name)+') WITH FULLSCAN' AS
VARCHAR(8000))
        WHEN 3
        THEN    CAST('CREATE STATISTICS '+QUOTENAME('stat_'+table_schema_name+ '_' + table_name +
'_'+column_name)+' ON
'+QUOTENAME(table_schema_name)+'.'+QUOTENAME(table_name)+'('+QUOTENAME(column_name)+') WITH SAMPLE
'+@sample_pct+'PERCENT' AS VARCHAR(8000))
        END AS create_stat_ddl
FROM T
;

DECLARE @i INT              = 1
,       @t INT              = (SELECT COUNT(*) FROM #stats_ddl)
,       @s NVARCHAR(4000)   = N''
;

WHILE @i <= @t
BEGIN
    SET @s=(SELECT create_stat_ddl FROM #stats_ddl WHERE seq_nmbr = @i);

    PRINT @s
    EXEC sp_executesql @s
    SET @i+=1;
END

DROP TABLE #stats_ddl;
```

To create statistics on all columns in the table with this procedure, simply call the procedure.

```
prc_sqldw_create_stats;
```

Examples: update statistics

To update statistics, you can:

1. Update one statistics object. Specify the name of the statistics object you wish to update.
2. Update all statistics objects on a table. Specify the name of the table instead of one specific statistics object.

A. Update one specific statistics object

Use the following syntax to update a specific statistics object:

```
UPDATE STATISTICS [schema_name].[table_name]([stat_name]);
```

.

```
UPDATE STATISTICS [dbo].[table1] ([stats_col1]);
```

By updating specific statistics objects, you can minimize the time and resources required to manage statistics. This requires some thought, though, to choose the best statistics objects to update.

B. Update all statistics on a table

This shows a simple method for updating all the statistics objects on a table.

```
UPDATE STATISTICS [schema_name].[table_name];
```

For example:

```
UPDATE STATISTICS dbo.table1;
```

This statement is easy to use. Just remember this updates all statistics on the table, and therefore might perform more work than is necessary. If the performance is not an issue, this is definitely the easiest and most complete way to guarantee statistics are up-to-date.

> **NOTE**
> When updating all statistics on a table, SQL Data Warehouse does a scan to sample the table for each statistics. If the table is large, has many columns, and many statistics, it might be more efficient to update individual statistics based on need.

For an implementation of an `UPDATE STATISTICS` procedure please see the Temporary Tables article. The implementation method is slightly different to the `CREATE STATISTICS` procedure above but the end result is the same.

For the full syntax, see Update Statistics on MSDN.

Statistics metadata

There are several system view and functions that you can use to find information about statistics. For example, you can see if a statistics object might be out-of-date by using the stats-date function to see when statistics were last created or updated.

Catalog views for statistics

These system views provide information about statistics:

CATALOG VIEW	DESCRIPTION
sys.columns	One row for each column.
sys.objects	One row for each object in the database.
sys.schemas	One row for each schema in the database.
sys.stats	One row for each statistics object.

CATALOG VIEW	DESCRIPTION
sys.stats_columns	One row for each column in the statistics object. Links back to sys.columns.
sys.tables	One row for each table (includes external tables).
sys.table_types	One row for each data type.

System functions for statistics

These system functions are useful for working with statistics:

SYSTEM FUNCTION	DESCRIPTION
STATS_DATE	Date the statistics object was last updated.
DBCC SHOW_STATISTICS	Provides summary level and detailed information about the distribution of values as understood by the statistics object.

Combine statistics columns and functions into one view

This view brings columns that relate to statistics, and results from the [STATS_DATE()][]function together.

```
CREATE VIEW dbo.vstats_columns
AS
SELECT
        sm.[name]                           AS [schema_name]
,       tb.[name]                           AS [table_name]
,       st.[name]                           AS [stats_name]
,       st.[filter_definition]              AS [stats_filter_defiinition]
,       st.[has_filter]                     AS [stats_is_filtered]
,       STATS_DATE(st.[object_id],st.[stats_id])
                                            AS [stats_last_updated_date]
,       co.[name]                           AS [stats_column_name]
,       ty.[name]                           AS [column_type]
,       co.[max_length]                     AS [column_max_length]
,       co.[precision]                      AS [column_precision]
,       co.[scale]                          AS [column_scale]
,       co.[is_nullable]                    AS [column_is_nullable]
,       co.[collation_name]                 AS [column_collation_name]
,       QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])
                                            AS two_part_name
,       QUOTENAME(DB_NAME())+'.'+QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])
                                            AS three_part_name
FROM    sys.objects                         AS ob
JOIN    sys.stats           AS st ON    ob.[object_id]      = st.[object_id]
JOIN    sys.stats_columns   AS sc ON    st.[stats_id]       = sc.[stats_id]
                            AND         st.[object_id]      = sc.[object_id]
JOIN    sys.columns         AS co ON    sc.[column_id]      = co.[column_id]
                            AND         sc.[object_id]      = co.[object_id]
JOIN    sys.types           AS ty ON    co.[user_type_id]   = ty.[user_type_id]
JOIN    sys.tables          AS tb ON  co.[object_id]        = tb.[object_id]
JOIN    sys.schemas         AS sm ON  tb.[schema_id]        = sm.[schema_id]
WHERE   1=1
AND     st.[user_created] = 1
;
```

DBCC SHOW_STATISTICS() examples

DBCC SHOW_STATISTICS() shows the data held within a statistics object. This data comes in three parts

2. Density Vector
3. Histogram

The header metadata about the statistics. The histogram displays the distribution of values in the first key column of the statistics object. The density vector measures cross-column correlation. SQLDW computes cardinality estimates with any of the data in the statistics object.

Show header, density, and histogram

This simple example shows all three parts of a statistics object.

```
DBCC SHOW_STATISTICS([<schema_name>.<table_name>],<stats_name>)
```

For example:

```
DBCC SHOW_STATISTICS (dbo.table1, stats_col1);
```

Show one or more parts of DBCC SHOW_STATISTICS();

If you are only interested in viewing specific parts, use the `WITH` clause and specify which parts you want to see:

```
DBCC SHOW_STATISTICS([<schema_name>.<table_name>],<stats_name>) WITH stat_header, histogram,
density_vector
```

For example:

```
DBCC SHOW_STATISTICS (dbo.table1, stats_col1) WITH histogram, density_vector
```

DBCC SHOW_STATISTICS() differences

DBCC SHOW_STATISTICS() is more strictly implemented in SQL Data Warehouse compared to SQL Server.

1. Undocumented features are not supported
2. Cannot use Stats_stream
3. Cannot join results for specific subsets of statistics data e.g. (STAT_HEADER JOIN DENSITY_VECTOR)
4. NO_INFOMSGS cannot be set for message suppression
5. Square brackets around statistics names cannot be used
6. Cannot use column names to identify statistics objects
7. Custom error 2767 is not supported

Next steps

For more details, see DBCC SHOW_STATISTICS on MSDN. To learn more, see the articles on Table Overview, Table Data Types, Distributing a Table, Indexing a Table, Partitioning a Table and Temporary Tables. For more about best practices, see SQL Data Warehouse Best Practices.

Temporary tables in SQL Data Warehouse

12/6/2017 • 5 min to read • Edit Online

Temporary tables are useful when processing data - especially during transformation where the intermediate results are transient. In SQL Data Warehouse, temporary tables exist at the session level. They are only visible to the session in which they were created and are automatically dropped when that session logs off. Temporary tables offer a performance benefit because their results are written to local rather than remote storage. Temporary tables are slightly different in Azure SQL Data Warehouse than Azure SQL Database as they can be accessed from anywhere inside the session, including both inside and outside of a stored procedure.

This article contains essential guidance for using temporary tables and highlights the principles of session level temporary tables. Using the information in this article can help you modularize your code, improving both reusability and ease of maintenance of your code.

Create a temporary table

Temporary tables are created by prefixing your table name with a `#`. For example:

```
CREATE TABLE #stats_ddl
(
    [schema_name]        NVARCHAR(128) NOT NULL
,    [table_name]            NVARCHAR(128) NOT NULL
,    [stats_name]            NVARCHAR(128) NOT NULL
,    [stats_is_filtered]     BIT           NOT NULL
,    [seq_nmbr]              BIGINT        NOT NULL
,    [two_part_name]         NVARCHAR(260) NOT NULL
,    [three_part_name]       NVARCHAR(400) NOT NULL
)
WITH
(
    DISTRIBUTION = HASH([seq_nmbr])
,    HEAP
)
```

Temporary tables can also be created with a `CTAS` using exactly the same approach:

```
CREATE TABLE #stats_ddl
WITH
(
    DISTRIBUTION = HASH([seq_nmbr])
,    HEAP
)
AS
(
SELECT
        sm.[name]                                                       AS [schema_name]
,        tb.[name]                                                       AS [table_name]
,        st.[name]                                                       AS [stats_name]
,        st.[has_filter]                                                   AS [stats_is_filtered]
,       ROW_NUMBER()
        OVER(ORDER BY (SELECT NULL))                                    AS [seq_nmbr]
,                                QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])  AS [two_part_name]
,        QUOTENAME(DB_NAME())+'.'+QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])  AS [three_part_name]
FROM    sys.objects            AS ob
JOIN    sys.stats            AS st    ON    ob.[object_id]        = st.[object_id]
JOIN    sys.stats_columns    AS sc    ON    st.[stats_id]        = sc.[stats_id]
                                    AND st.[object_id]        = sc.[object_id]
JOIN    sys.columns            AS co    ON    sc.[column_id]        = co.[column_id]
                                    AND    sc.[object_id]        = co.[object_id]
JOIN    sys.tables            AS tb    ON    co.[object_id]        = tb.[object_id]
JOIN    sys.schemas            AS sm    ON    tb.[schema_id]        = sm.[schema_id]
WHERE    1=1
AND        st.[user_created]   = 1
GROUP BY
        sm.[name]
,        tb.[name]
,        st.[name]
,        st.[filter_definition]
,        st.[has_filter]
)
SELECT
    CASE @update_type
    WHEN 1
    THEN 'UPDATE STATISTICS '+[two_part_name]+'('+[stats_name]+');'
    WHEN 2
    THEN 'UPDATE STATISTICS '+[two_part_name]+'('+[stats_name]+') WITH FULLSCAN;'
    WHEN 3
    THEN 'UPDATE STATISTICS '+[two_part_name]+'('+[stats_name]+') WITH SAMPLE '+CAST(@sample_pct AS
VARCHAR(20))+' PERCENT;'
    WHEN 4
    THEN 'UPDATE STATISTICS '+[two_part_name]+'('+[stats_name]+') WITH RESAMPLE;'
    END AS [update_stats_ddl]
,   [seq_nmbr]
FROM    t1
;
```

NOTE

`CTAS` is a powerful command and has the added advantage of being efficient in its use of transaction log space.

Dropping temporary tables

When a new session is created, no temporary tables should exist. However, if you are calling the same stored procedure, which creates a temporary with the same name, to ensure that your `CREATE TABLE` statements are successful a simple pre-existence check with a `DROP` can be used as in the following example:

```
IF OBJECT_ID('tempdb..#stats_ddl') IS NOT NULL
BEGIN
    DROP TABLE #stats_ddl
END
```

For coding consistency, it is a good practice to use this pattern for both tables and temporary tables. It is also a good idea to use `DROP TABLE` to remove temporary tables when you have finished with them in your code. In stored procedure development, it is common to see the drop commands bundled together at the end of a procedure to ensure these objects are cleaned up.

```
DROP TABLE #stats_ddl
```

Modularizing code

Since temporary tables can be seen anywhere in a user session, this can be exploited to help you modularize your application code. For example, the following stored procedure generates DDL to update all statistics in the database by statistic name.

```
CREATE PROCEDURE    [dbo].[prc_sqldw_update_stats]
(   @update_type    tinyint -- 1 default 2 fullscan 3 sample 4 resample
    ,@sample_pct     tinyint
)
AS

IF @update_type NOT IN (1,2,3,4)
BEGIN;
    THROW 151000,'Invalid value for @update_type parameter. Valid range 1 (default), 2 (fullscan), 3
(sample) or 4 (resample).',1;
END;

IF @sample_pct IS NULL
BEGIN;
    SET @sample_pct = 20;
END;

IF OBJECT_ID('tempdb..#stats_ddl') IS NOT NULL
BEGIN
    DROP TABLE #stats_ddl
END

CREATE TABLE #stats_ddl
WITH
(
    DISTRIBUTION = HASH([seq_nmbr])
)
AS
(
SELECT
        sm.[name]                                                                   AS [schema_name]
,        tb.[name]                                                                   AS [table_name]
,        st.[name]                                                                   AS [stats_name]
,        st.[has_filter]                                                               AS [stats_is_filtered]
,       ROW_NUMBER()
        OVER(ORDER BY (SELECT NULL))                                                AS [seq_nmbr]
,                               QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])  AS [two_part_name]
,        QUOTENAME(DB_NAME())+'.'+QUOTENAME(sm.[name])+'.'+QUOTENAME(tb.[name])  AS [three_part_name]
FROM    sys.objects           AS ob
JOIN    sys.stats           AS st    ON    ob.[object_id]        = st.[object_id]
JOIN    sys.stats_columns    AS sc    ON    st.[stats_id]        = sc.[stats_id]
                                AND st.[object_id]        = sc.[object_id]
JOIN    sys.columns           AS co    ON    co.[column_id]          co.[column_id]
```

```
JOIN    sys.tables            AS tb    ON    co.[object_id]        = tb.[object_id]
JOIN    sys.schemas           AS sm    ON    tb.[schema_id]        = sm.[schema_id]
WHERE    1=1
AND        st.[user_created]   = 1
GROUP BY
        sm.[name]
,        tb.[name]
,        st.[name]
,        st.[filter_definition]
,        st.[has_filter]
)
SELECT
    CASE @update_type
    WHEN 1
    THEN 'UPDATE STATISTICS '+[two_part_name]+'('+[stats_name]+');'
    WHEN 2
    THEN 'UPDATE STATISTICS '+[two_part_name]+'('+[stats_name]+') WITH FULLSCAN;'
    WHEN 3
    THEN 'UPDATE STATISTICS '+[two_part_name]+'('+[stats_name]+') WITH SAMPLE '+CAST(@sample_pct AS
VARCHAR(20))+' PERCENT;'
    WHEN 4
    THEN 'UPDATE STATISTICS '+[two_part_name]+'('+[stats_name]+') WITH RESAMPLE;'
    END AS [update_stats_ddl]
,   [seq_nmbr]
FROM    t1
;
GO
```

At this stage, the only action that has occurred is the creation of a stored procedure that generatess a temporary table, #stats_ddl, with DDL statements. This stored procedure drops #stats_ddl if it already exists to ensure it does not fail if run more than once within a session. However, since there is no `DROP TABLE` at the end of the stored procedure, when the stored procedure completes, it leaves the created table so that it can be read outside of the stored procedure. In SQL Data Warehouse, unlike other SQL Server databases, it is possible to use the temporary table outside of the procedure that created it. SQL Data Warehouse temporary tables can be used **anywhere** inside the session. This can lead to more modular and manageable code as in the following example:

```
EXEC [dbo].[prc_sqldw_update_stats] @update_type = 1, @sample_pct = NULL;

DECLARE @i INT              = 1
,       @t INT              = (SELECT COUNT(*) FROM #stats_ddl)
,       @s NVARCHAR(4000)   = N''

WHILE @i <= @t
BEGIN
    SET @s=(SELECT update_stats_ddl FROM #stats_ddl WHERE seq_nmbr = @i);

    PRINT @s
    EXEC sp_executesql @s
    SET @i+=1;
END

DROP TABLE #stats_ddl;
```

Temporary table limitations

SQL Data Warehouse does impose a couple of limitations when implementing temporary tables. Currently, only session scoped temporary tables are supported. Global Temporary Tables are not supported. In addition, views cannot be created on temporary tables.

Next steps

To learn more, see the articles on Table Overview, Table Data Types, Distributing a Table, Indexing a Table, Partitioning a Table and Maintaining Table Statistics. For more about best practices, see SQL Data Warehouse Best Practices.

Dynamic SQL in SQL Data Warehouse

6/27/2017 • 1 min to read • Edit Online

When developing application code for SQL Data Warehouse you may need to use dynamic sql to help deliver flexible, generic and modular solutions. SQL Data Warehouse does not support blob data types at this time. This may limit the size of your strings as blob types include both varchar(max) and nvarchar(max) types. If you have used these types in your application code when building very large strings, you will need to break the code into chunks and use the EXEC statement instead.

A simple example:

```
DECLARE @sql_fragment1 VARCHAR(8000)=' SELECT name '
,       @sql_fragment2 VARCHAR(8000)=' FROM sys.system_views '
,       @sql_fragment3 VARCHAR(8000)=' WHERE name like ''%table%''';

EXEC( @sql_fragment1 + @sql_fragment2 + @sql_fragment3);
```

If the string is short you can use sp_executesql as normal.

> **NOTE**
> Statements executed as dynamic SQL will still be subject to all TSQL validation rules.

Next steps

For more development tips, see development overview.

Group by options in SQL Data Warehouse

6/27/2017 • 3 min to read • Edit Online

The GROUP BY clause is used to aggregate data to a summary set of rows. It also has a few options that extend it's functionality that need to be worked around as they are not directly supported by Azure SQL Data Warehouse.

These options are

- GROUP BY with ROLLUP
- GROUPING SETS
- GROUP BY with CUBE

Rollup and grouping sets options

The simplest option here is to use `UNION ALL` instead to perform the rollup rather than relying on the explicit syntax. The result is exactly the same

Below is an example of a group by statement using the `ROLLUP` option:

```
SELECT [SalesTerritoryCountry]
,      [SalesTerritoryRegion]
,      SUM(SalesAmount)            AS TotalSalesAmount
FROM  dbo.factInternetSales s
JOIN  dbo.DimSalesTerritory t      ON s.SalesTerritoryKey       = t.SalesTerritoryKey
GROUP BY ROLLUP (
                        [SalesTerritoryCountry]
                ,       [SalesTerritoryRegion]
                )
;
```

By using ROLLUP we have requested the following aggregations:

- Country and Region
- Country
- Grand Total

To replace this you will need to use `UNION ALL`; specifying the aggregations required explicitly to return the same results:

```
SELECT [SalesTerritoryCountry]
,      [SalesTerritoryRegion]
,      SUM(SalesAmount) AS TotalSalesAmount
FROM  dbo.factInternetSales s
JOIN  dbo.DimSalesTerritory t     ON s.SalesTerritoryKey       = t.SalesTerritoryKey
GROUP BY
       [SalesTerritoryCountry]
,      [SalesTerritoryRegion]
UNION ALL
SELECT [SalesTerritoryCountry]
,      NULL
,      SUM(SalesAmount) AS TotalSalesAmount
FROM  dbo.factInternetSales s
JOIN  dbo.DimSalesTerritory t     ON s.SalesTerritoryKey       = t.SalesTerritoryKey
GROUP BY
       [SalesTerritoryCountry]
UNION ALL
SELECT NULL
,      NULL
,      SUM(SalesAmount) AS TotalSalesAmount
FROM  dbo.factInternetSales s
JOIN  dbo.DimSalesTerritory t     ON s.SalesTerritoryKey       = t.SalesTerritoryKey;
```

For GROUPING SETS all we need to do is adopt the same principal but only create UNION ALL sections for the aggregation levels we want to see

Cube options

It is possible to create a GROUP BY WITH CUBE using the UNION ALL approach. The problem is that the code can quickly become cumbersome and unwieldy. To mitigate this you can use this more advanced approach.

Let's use the example above.

The first step is to define the 'cube' that defines all the levels of aggregation that we want to create. It is important to take note of the CROSS JOIN of the two derived tables. This generates all the levels for us. The rest of the code is really there for formatting.

```
CREATE TABLE #Cube
WITH
(   DISTRIBUTION = ROUND_ROBIN
,   LOCATION = USER_DB
)
AS
WITH GrpCube AS
(SELECT    CAST(ISNULL(Country,'NULL')+','+ISNULL(Region,'NULL') AS NVARCHAR(50)) as 'Cols'
,          CAST(ISNULL(Country+',','')+ISNULL(Region,'') AS NVARCHAR(50))  as 'GroupBy'
,          ROW_NUMBER() OVER (ORDER BY Country) as 'Seq'
FROM       ( SELECT 'SalesTerritoryCountry' as Country
             UNION ALL
             SELECT NULL
           ) c
CROSS JOIN ( SELECT 'SalesTerritoryRegion' as Region
             UNION ALL
             SELECT NULL
           ) r
)
SELECT Cols
,      CASE WHEN SUBSTRING(GroupBy,LEN(GroupBy),1) = ','
            THEN SUBSTRING(GroupBy,1,LEN(GroupBy)-1)
            ELSE GroupBy
       END AS GroupBy  --Remove Trailing Comma
,Seq
FROM GrpCube;
```

The results of the CTAS can be seen below:

	Cols	GroupBy	Seq
1	NULL,SalesTerritoryRegion	SalesTerritoryRegion	1
2	NULL,NULL		2
3	SalesTerritoryCountry,SalesTerritoryRegion	SalesTerritoryCountry,SalesTerritoryRegion	3
4	SalesTerritoryCountry,NULL	SalesTerritoryCountry	4

The second step is to specify a target table to store interim results:

```
DECLARE
 @SQL NVARCHAR(4000)
,@Columns NVARCHAR(4000)
,@GroupBy NVARCHAR(4000)
,@i INT = 1
,@nbr INT = 0
;
CREATE TABLE #Results
(
 [SalesTerritoryCountry] NVARCHAR(50)
,[SalesTerritoryRegion]  NVARCHAR(50)
,[TotalSalesAmount]      MONEY
)
WITH
(   DISTRIBUTION = ROUND_ROBIN
,   LOCATION = USER_DB
)
;
```

The third step is to loop over our cube of columns performing the aggregation. The query will run once for every row in the #Cube temporary table and store the results in the #Results temp table

```
SET @nbr =(SELECT MAX(Seq) FROM #Cube);

WHILE @i<=@nbr
BEGIN
    SET @Columns = (SELECT Cols    FROM #Cube where seq = @i);
    SET @GroupBy = (SELECT GroupBy FROM #Cube where seq = @i);

    SET @SQL ='INSERT INTO #Results
              SELECT '+@Columns+'
              ,      SUM(SalesAmount) AS TotalSalesAmount
              FROM  dbo.factInternetSales s
              JOIN  dbo.DimSalesTerritory t
              ON s.SalesTerritoryKey = t.SalesTerritoryKey
              '+CASE WHEN @GroupBy <>''
                     THEN 'GROUP BY '+@GroupBy ELSE '' END

    EXEC sp_executesql @SQL;
    SET @i +=1;
END
```

Lastly we can return the results by simply reading from the #Results temporary table

```
SELECT *
FROM #Results
ORDER BY 1,2,3
;
```

By breaking the code up into sections and generating a looping construct the code becomes more manageable and maintainable.

Next steps

For more development tips, see development overview.

Use labels to instrument queries in SQL Data Warehouse

6/27/2017 • 1 min to read • Edit Online

SQL Data Warehouse supports a concept called query labels. Before going into any depth let's look at an example of one:

```
SELECT *
FROM sys.tables
OPTION (LABEL = 'My Query Label')
;
```

This last line tags the string 'My Query Label' to the query. This is particularly helpful as the label is query-able through the DMVs. This provides us with a mechanism to track down problem queries and also to help identify progress through an ETL run.

A good naming convention really helps here. For example something like ' PROJECT : PROCEDURE : STATEMENT : COMMENT' would help to uniquely identify the query in amongst all the code in source control.

To search by label you can use the following query that uses the dynamic management views:

```
SELECT  *
FROM    sys.dm_pdw_exec_requests r
WHERE   r.[label] = 'My Query Label'
;
```

NOTE

It is essential that you wrap square brackets or double quotes around the word label when querying. Label is a reserved word and will caused an error if it has not been delimited.

Next steps

For more development tips, see development overview.

Loops in SQL Data Warehouse

6/27/2017 • 1 min to read • Edit Online

SQL Data Warehouse supports the WHILE loop for repeatedly executing statement blocks. This will continue for as long as the specified conditions are true or until the code specifically terminates the loop using the `BREAK` keyword. Loops are particularly useful for replacing cursors defined in SQL code. Fortunately, almost all cursors that are written in SQL code are of the fast forward, read only variety. Therefore WHILE loops are a great alternative if you find yourself having to replace one.

Leveraging loops and replacing cursors in SQL Data Warehouse

However, before diving in head first you should ask yourself the following question: "Could this cursor be re-written to use set based operations?". In many cases the answer will be yes and is often the best approach. A set based operation often performs significantly faster than an iterative, row by row approach.

Fast forward read-only cursors can be easily replaced with a looping construct. Below is a simple example. This code example updates the statistics for every table in the database. By iterating over the tables in the loop we are able to execute each command in sequence.

First, create a temporary table containing a unique row number used to identify the individual statements:

```
CREATE TABLE #tbl
WITH
( DISTRIBUTION = ROUND_ROBIN
)
AS
SELECT  ROW_NUMBER() OVER(ORDER BY (SELECT NULL)) AS Sequence
,       [name]
,       'UPDATE STATISTICS '+QUOTENAME([name]) AS sql_code
FROM    sys.tables
;
```

Second, initialize the variables required to perform the loop:

```
DECLARE @nbr_statements INT = (SELECT COUNT(*) FROM #tbl)
,       @i INT = 1
;
```

Now loop over statements executing them one at a time:

```
WHILE   @i <= @nbr_statements
BEGIN
    DECLARE @sql_code NVARCHAR(4000) = (SELECT sql_code FROM #tbl WHERE Sequence = @i);
    EXEC    sp_executesql @sql_code;
    SET     @i +=1;
END
```

Finally drop the temporary table created in the first step

```
DROP TABLE #tbl;
```

Next steps

For more development tips, see development overview.

Stored procedures in SQL Data Warehouse

6/27/2017 • 2 min to read • Edit Online

SQL Data Warehouse supports many of the Transact-SQL features found in SQL Server. More importantly there are scale out specific features that we will want to leverage to maximize the performance of your solution.

However, to maintain the scale and performance of SQL Data Warehouse there are also some features and functionality that have behavioral differences and others that are not supported.

This article explains how to implement stored procedures within SQL Data Warehouse.

Introducing stored procedures

Stored procedures are a great way for encapsulating your SQL code; storing it close to your data in the data warehouse. By encapsulating the code into manageable units stored procedures help developers modularize their solutions; facilitating greater re-usability of code. Each stored procedure can also accept parameters to make them even more flexible.

SQL Data Warehouse provides a simplified and streamlined stored procedure implementation. The biggest difference compared to SQL Server is that the stored procedure is not pre-compiled code. In data warehouses we are generally less concerned with the compilation time. It is more important that the stored procedure code is correctly optimised when operating against large data volumes. The goal is to save hours, minutes and seconds not milliseconds. It is therefore more helpful to think of stored procedures as containers for SQL logic.

When SQL Data Warehouse executes your stored procedure the SQL statements are parsed, translated and optimized at run time. During this process each statement is converted into distributed queries. The SQL code that is actually executed against the data is different to the query submitted.

Nesting stored procedures

When stored procedures call other stored procedures or execute dynamic sql then the inner stored procedure or code invocation is said to be nested.

SQL Data Warehouse support a maximum of 8 nesting levels. This is slightly different to SQL Server. The nest level in SQL Server is 32.

The top level stored procedure call equates to nest level 1

```
EXEC prc_nesting
```

If the stored procedure also makes another EXEC call then this will increase the nest level to 2

```
CREATE PROCEDURE prc_nesting
AS
EXEC prc_nesting_2  -- This call is nest level 2
GO
EXEC prc_nesting
```

If the second procedure then executes some dynamic sql then this will increase the nest level to 3

```
CREATE PROCEDURE prc_nesting_2
AS
EXEC sp_executesql 'SELECT 'another nest level'  -- This call is nest level 2
GO
EXEC prc_nesting
```

Note SQL Data Warehouse does not currently support @@NESTLEVEL. You will need to keep a track of your nest level yourself. It is unlikely you will hit the 8 nest level limit but if you do you will need to re-work your code and "flatten" it so that it fits within this limit.

INSERT..EXECUTE

SQL Data Warehouse does not permit you to consume the result set of a stored procedure with an INSERT statement. However, there is an alternative approach you can use.

Please refer to the following article on temporary tables for an example on how to do this.

Limitations

There are some aspects of Transact-SQL stored procedures that are not implemented in SQL Data Warehouse.

They are:

- temporary stored procedures
- numbered stored procedures
- extended stored procedures
- CLR stored procedures
- encryption option
- replication option
- table-valued parameters
- read-only parameters
- default parameters
- execution contexts
- return statement

Next steps

For more development tips, see development overview.

Transactions in SQL Data Warehouse

6/27/2017 • 5 min to read • Edit Online

As you would expect, SQL Data Warehouse supports transactions as part of the data warehouse workload. However, to ensure the performance of SQL Data Warehouse is maintained at scale some features are limited when compared to SQL Server. This article highlights the differences and lists the others.

Transaction isolation levels

SQL Data Warehouse implements ACID transactions. However, the Isolation of the transactional support is limited to `READ UNCOMMITTED` and this cannot be changed. You can implement a number of coding methods to prevent dirty reads of data if this is a concern for you. The most popular methods leverage both CTAS and table partition switching (often known as the sliding window pattern) to prevent users from querying data that is still being prepared. Views that pre-filter the data is also a popular approach.

Transaction size

A single data modification transaction is limited in size. The limit today is applied "per distribution". Therefore, the total allocation can be calculated by multiplying the limit by the distribution count. To approximate the maximum number of rows in the transaction divide the distribution cap by the total size of each row. For variable length columns consider taking an average column length rather than using the maximum size.

In the table below the following assumptions have been made:

- An even distribution of data has occurred
- The average row length is 250 bytes

DWU	CAP PER DISTRIBUTION (GIB)	NUMBER OF DISTRIBUTIONS	MAX TRANSACTION SIZE (GIB)	# ROWS PER DISTRIBUTION	MAX ROWS PER TRANSACTION
DW100	1	60	60	4,000,000	240,000,000
DW200	1.5	60	90	6,000,000	360,000,000
DW300	2.25	60	135	9,000,000	540,000,000
DW400	3	60	180	12,000,000	720,000,000
DW500	3.75	60	225	15,000,000	900,000,000
DW600	4.5	60	270	18,000,000	1,080,000,000
DW1000	7.5	60	450	30,000,000	1,800,000,000
DW1200	9	60	540	36,000,000	2,160,000,000
DW1500	11.25	60	675	45,000,000	2,700,000,000
DW2000	15	60	900	60,000,000	3,600,000,000

DWU	CAP PER DISTRIBUTION (GIB)	NUMBER OF DISTRIBUTIONS	MAX TRANSACTION SIZE (GIB)	# ROWS PER DISTRIBUTION	MAX ROWS PER TRANSACTION
DW3000	22.5	60	1,350	90,000,000	5,400,000,000
DW6000	45	60	2,700	180,000,000	10,800,000,000

The transaction size limit is applied per transaction or operation. It is not applied across all concurrent transactions. Therefore each transaction is permitted to write this amount of data to the log.

To optimize and minimize the amount of data written to the log please refer to the Transactions best practices article.

WARNING

The maximum transaction size can only be achieved for HASH or ROUND_ROBIN distributed tables where the spread of the data is even. If the transaction is writing data in a skewed fashion to the distributions then the limit is likely to be reached prior to the maximum transaction size.

Transaction state

SQL Data Warehouse uses the XACT_STATE() function to report a failed transaction using the value -2. This means that the transaction has failed and is marked for rollback only

NOTE

The use of -2 by the XACT_STATE function to denote a failed transaction represents different behavior to SQL Server. SQL Server uses the value -1 to represent an un-committable transaction. SQL Server can tolerate some errors inside a transaction without it having to be marked as un-committable. For example `SELECT 1/0` would cause an error but not force a transaction into an un-committable state. SQL Server also permits reads in the un-committable transaction. However, SQL Data Warehouse does not let you do this. If an error occurs inside a SQL Data Warehouse transaction it will automatically enter the -2 state and you will not be able to make any further select statements until the statement has been rolled back. It is therefore important to check that your application code to see if it uses XACT_STATE() as you may need to make code modifications.

For example, in SQL Server you might see a transaction that looks like this: