# [Policy-based management](https://docs.microsoft.com/en-us/sql/relational-databases/policy-based-management/administer-servers-by-using-policy-based-management)

Policy-Based Management is a policy based system for managing one or more instances of SQL Server. Use is to create conditions that contain condition expressions. Then, create policies that apply the conditions to database target objects.

For example, as the database administrator, you may want to ensure that certain servers do not have Database Mail enabled, so you create a condition and a policy that sets that server option.

**IMPORTANT!!** Policies can affect how some features work. For example, change data capture and transactional replication both use the systranschemas table, which does not have an index. If you enable a policy that all tables must have an index, enforcing compliance of the policy will cause these features to fail.

USE msdb;

GO

EXEC dbo.sp\_syspolicy\_configure @name = N'Enabled', @value = 1;

## Three Policy-Based Management components

Policy-Based Management has three components:

* Policy management. Policy administrators create policies.
* Explicit administration. Administrators select one or more managed targets and explicitly check that the targets comply with a specific policy, or explicitly make the targets comply with a policy.
* Evaluation modes. There are four evaluation modes; three can be automated:
  + **On demand**. This mode evaluates the policy when directly specified by the user.
  + **On change: prevent**. This automated mode uses DDL triggers to prevent policy violations.

**IMPORTANT!** If the nested triggers server configuration option is disabled, **On change: prevent** will not work correctly. Policy-Based Management relies on DDL triggers to detect and roll back DDL operations that do not comply with policies that use this evaluation mode. Removing the Policy-Based Management DDL triggers or disabling nest triggers, will cause this evaluation mode to fail or perform unexpectedly.

* + **On change: log only**. This automated mode uses event notification to evaluate a policy when a relevant change is made.
  + **On schedule**. This automated mode uses a SQL Server Agent job to periodically evaluate a policy.

When automated policies are not enabled, Policy-Based Management will not affect system performance.

## Terms

**Policy-Based Management managed target** Entities that are managed by Policy-Based Management, such as an instance of the SQL Server Database Engine, a database, a table, or an index. All targets in a server instance form a target hierarchy. A target set is the set of targets that results from applying a set of target filters to the target hierarchy, for example, all the tables in the database owned by the HumanResources schema.

**Policy-Based Management facet** A set of logical properties that model the behavior or characteristics for certain types of managed targets. The number and characteristics of the properties are built into the facet and can be added or removed by only the maker of the facet. A target type can implement one or more management facets, and a management facet can be implemented by one or more target types. Some properties of a facet can only apply to a specific version..

**Policy-Based Management condition**  
A Boolean expression that specifies a set of allowed states of a Policy-Based Management managed target with regard to a management facet. SQL Server tries to observe collations when evaluating a condition. When SQL Server collations do not exactly match Windows collations, test your condition to determine how the algorithm resolves conflicts.+

**Policy-Based Management policy**  
A Policy-Based Management condition and the expected behavior, for example, evaluation mode, target filters, and schedule. A policy can contain only one condition. Policies can be enabled or disabled. Policies are stored in the msdb database.

#### To view a policy's properties

USE msdb;

GO

SELECT name, execution\_mode, description, is\_enabled,

job\_id FROM syspolicy\_policies;

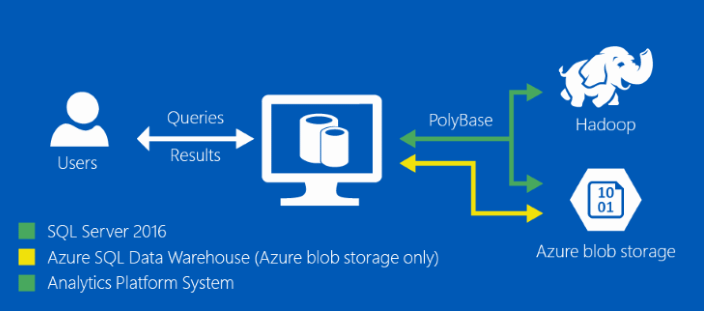
#### To view a condition's properties

SELECT name, description, facet, expression, s\_name\_condition,

obj\_name FROM syspolicy\_conditions;

# [PolyBase](https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-guide)

PolyBase is a technology that accesses data outside of the database via the t-sql language. In SQL Server 2016, it allows you to run queries on external data in Hadoop or to import/export data from Azure Blob Storage. Queries are optimized to push computation to Hadoop. In Azure SQL Data Warehouse, you can import/export data from Azure Blob Storage and Azure Data Lake Store.



## Why use PolyBase?

To make good decisions, you want to analyze both relational data and other data that is not structured into tables —notably Hadoop. This is difficult to do unless you have a way to transfer data among the different types of data stores. PolyBase bridges this gap by operating on data that is external to SQL Server.

To keep it simple, PolyBase does not require you to install additional software to your Hadoop environment. Querying external data uses the same syntax as querying a database table. This all happens transparently. PolyBase handles all the details behind-the-scenes, and no knowledge about Hadoop is required by the end user to query external tables.

PolyBase can:

* **Query data stored in Hadoop from SQL Server or PDW.** Users are storing data in cost-effective distributed and scalable systems, such as Hadoop. PolyBase makes it easy to query the data by using T-SQL.
* **Query data stored in Azure Blob Storage.** Azure blob storage is a convenient place to store data for use by Azure services. PolyBase makes it easy to access the data by using T-SQL.
* **Import data from Hadoop, Azure Blob Storage, or Azure Data Lake Store** Leverage the speed of Microsoft SQL's columnstore technology and analysis capabilities by importing data from Hadoop, Azure Blob Storage, or Azure Data Lake Store into relational tables. There is no need for a separate ETL or import tool.
* **Export data to Hadoop, Azure Blob Storage, or Azure Data Lake Store.** Archive data to Hadoop, Azure Blob Storage, or Azure Data Lake Store to achieve cost-effective storage and keep it online for easy access.
* **Integrate with BI tools.** Use PolyBase with Microsoft’s business intelligence and analysis stack, or use any third party tools that are compatible with SQL Server.

-- Values map to various external data sources.

-- Example: value 7 stands for Azure blob storage and Hortonworks HDP 2.3 on Linux.

sp\_configure @configname = 'hadoop connectivity', @configvalue = 7;

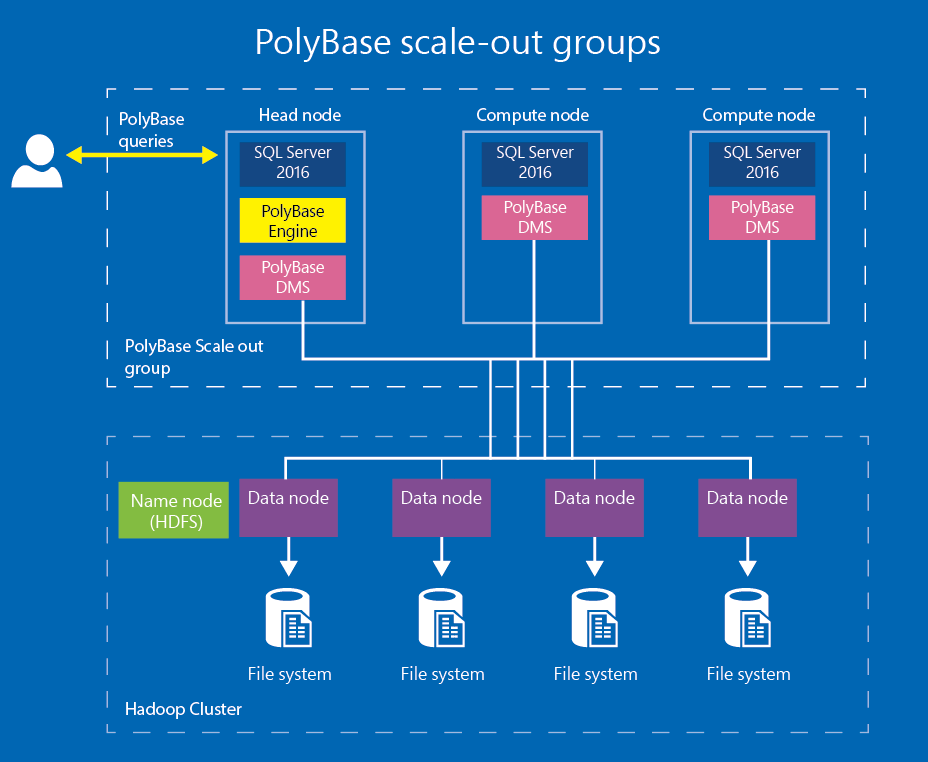
GO

RECONFIGURE

GO

## PolyBase scale-out groups

A standalone SQL Server instance with PolyBase can become a performance bottleneck when dealing with massive data sets in Hadoop or Azure Blob Storage. The PolyBase Group feature allows you to create a cluster of SQL Server instances to process large data sets from external data sources, such as Hadoop or Azure Blob Storage, in a scale-out fashion for better query performance.



### Heead node

The head node contains the SQL Server instance to which PolyBase queries are submitted. Each PolyBase group can have only one head node. A head node is a logical group of SQL Database Engine, PolyBase Engine and PolyBase Data Movement Service on the SQL Server instance.

### Compute node

A compute node contains the SQL Server instance that assists with scale-out query processing on external data. A compute node is a logical group of SQL Server and the PolyBase data movement service on the SQL Server instance. A PolyBase group can have multiple compute nodes.

### Distributed query processing

PolyBase queries are submitted to the SQL Server on the head node. The part of the query that refers to external tables is handed-off to the PolyBase engine.

The PolyBase engine is the key component behind PolyBase queries. It parses the query on external data, generates the query plan and distributes the work to the data movement service on the compute nodes for execution. After completion of the work, it receives the results from the compute nodes and submits them to SQL Server for processing and returning to the client.

The PolyBase data movement service receives instructions from the PolyBase engine and transfers data between HDFS and SQL Server, and between SQL Server instances on the head and compute nodes.

PolyBase in SQL Server 2016 only supports Windows users. If you try to use a SQL user to query a PolyBase external table, the query will fail.

-- Create an external data source.

-- LOCATION (Required) : Hadoop Name Node IP address and port.

-- RESOURCE MANAGER LOCATION (Optional): Hadoop Resource Manager location to enable pushdown computation.

-- CREDENTIAL (Optional): the database scoped credential, created above.

CREATE EXTERNAL DATA SOURCE MyHadoopCluster WITH (

TYPE = HADOOP,

LOCATION ='hdfs://10.xxx.xx.xxx:xxxx',

RESOURCE\_MANAGER\_LOCATION = '10.xxx.xx.xxx:xxxx',

CREDENTIAL = HadoopUser1

);

-- Create an external table pointing to data stored in Hadoop.

-- LOCATION: path to file or directory that contains the data (relative to HDFS root).

CREATE EXTERNAL TABLE [dbo].[CarSensor\_Data] (

[SensorKey] int NOT NULL,

[CustomerKey] int NOT NULL,

[GeographyKey] int NULL,

[Speed] float NOT NULL,

[YearMeasured] int NOT NULL

)

WITH (LOCATION='/Demo/',

DATA\_SOURCE = MyHadoopCluster,

FILE\_FORMAT = TextFileFormat

);