

Domain 3: Monitor and optimize operational resources in Azure SQL

Describe performance monitoring

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/describe-performance-monitoring/1-introduction>

Introduction

Completed

A major part of the job of a database administrator is proper performance monitoring. This task doesn't change when moving to a cloud platform. While Azure offers tools for monitoring, you may lack some specific controls around hardware that you would have in an on-premises environment which makes understanding how to identify and resolve performance bottlenecks while in Azure SQL that is much more critical.

Learning objectives

- Understanding methods to review potential performance issues
- Identify critical Azure metrics
- Learn how to collect metrics for an established baseline
- Use extended events for performance analysis
- Understand database watcher

2. Describe performance monitoring tools

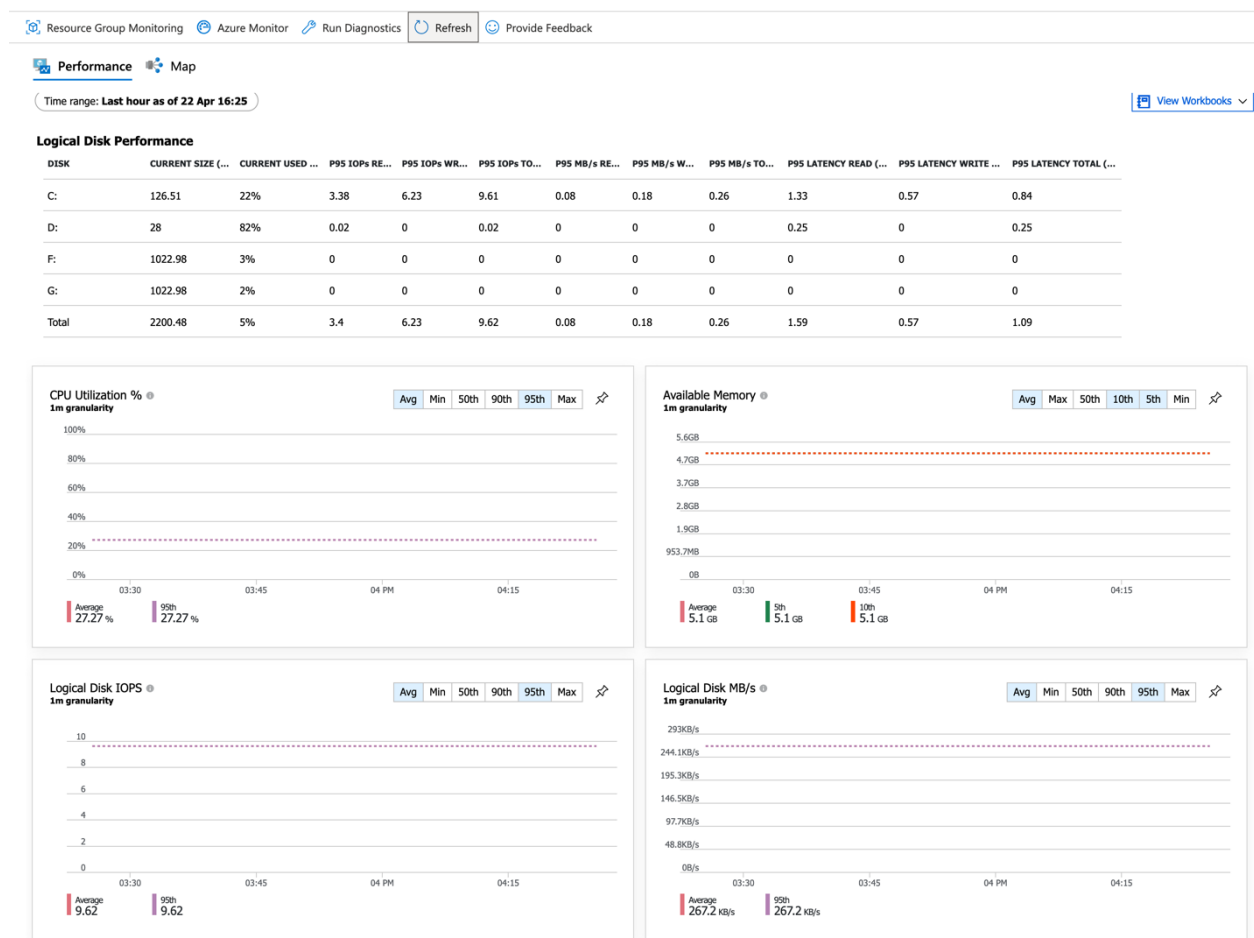
<https://learn.microsoft.com/en-us/training/modules/describe-performance-monitoring/2-describe-performance-monitoring-tools>

Describe performance monitoring tools

Completed

Azure provides multiple methods to monitor the performance of your resources and create a baseline. Each method can be tailored for a specific metric. The metrics that you can monitor vary depending on the type of Azure resource you're monitoring. For example, Azure SQL Database and SQL Server on an Azure Virtual Machine have different metrics available in the Azure portal.

The following examples are focused on an Azure Virtual Machine. When you deploy an Azure Virtual Machine from Azure Marketplace, an agent is installed in the virtual machine that provides a basic set of operating system metrics that are presented to you in the Azure portal. This agent supplies metrics to a service called Azure Monitor, which is a comprehensive platform monitoring solution that collects and displays a standard set of metrics from Azure resources. If you're using a virtual machine, the default metrics captured include CPU usage, network utilization, and disk read and write operations. You can capture extra metrics beyond what is captured in Azure monitor by enabling Monitoring Insights for your virtual machine as shown in the following image.



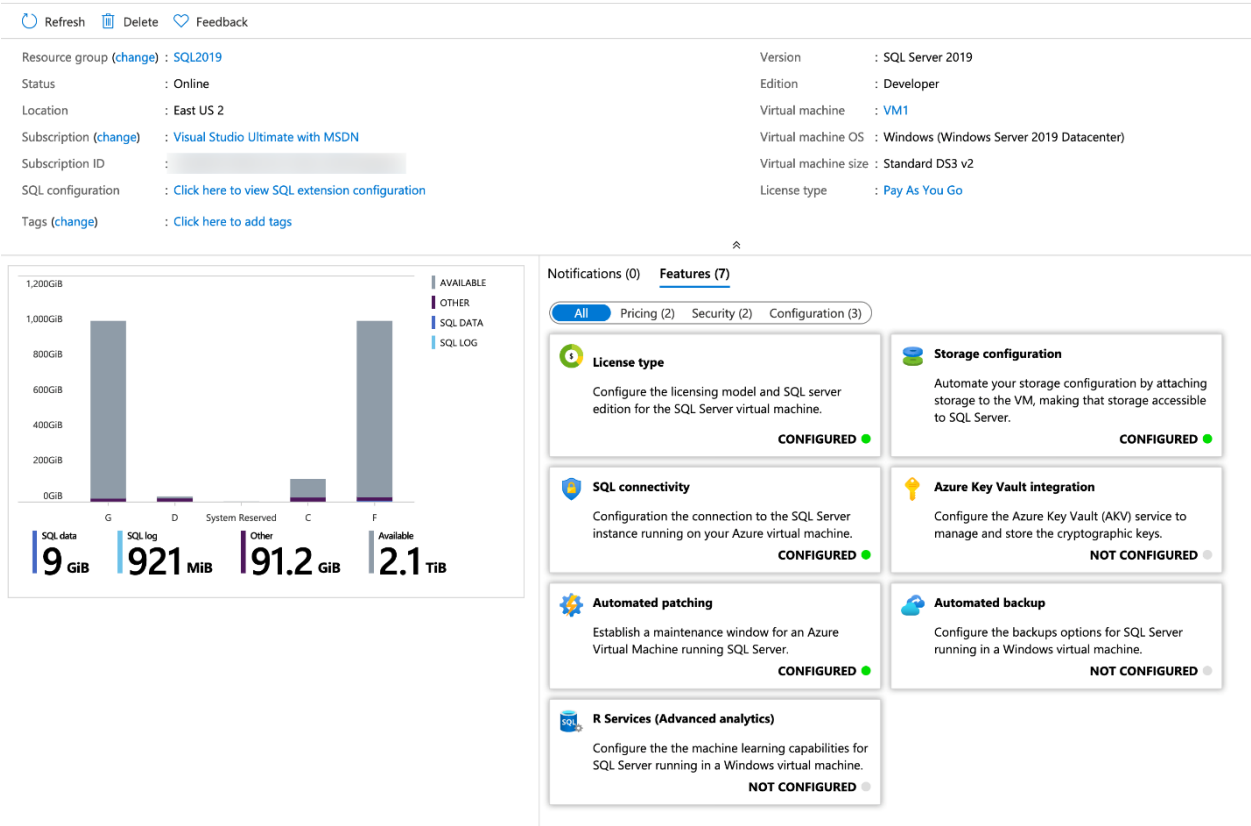
These metrics pertain to the operating system, not SQL Server. Notice that the namespace for each metric is the virtual machine host, not SQL Server.

You're unable to view SQL Server-specific metrics from within the portal. For detailed SQL Server-specific metrics, you need to gather them from the virtual machine itself.

Azure Monitoring Insights allows you to collect extra data points like storage latency, available memory, and disk capacity. These Azure Monitor Insights can be one way of viewing a baseline of performance for your Azure Virtual Machine including I/O performance, memory, and CPU utilization.

This data is stored in an Azure Log Analytics workspace. Azure Log Analytics is the primary tool in Azure for storing and querying log files of all kinds. Log Analytics is queried by a SQL-like language called Kusto Query Language (KQL).

If you create a virtual machine with one of the preconfigured SQL Server images in Azure Marketplace, you can also get the SQL virtual machine resource provider as shown in the following image.



You can launch this screen in the Azure portal by navigating to the **Settings** section of the main blade for an Azure Virtual Machine, and then selecting the **SQL Server configuration** option.



Virtual machine

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Settings

Networking

Connect

Disks

Size

Security

Advisor recommendations

Extensions

Continuous delivery

Availability + scaling

Configuration

Identity

SQL Server configuration

Properties

Locks

Operations

Bastion

Auto-shutdown

SQL management experience on Virtual Machines

The new SQL focused management experience provides a single view of all your [Virtual Machines running SQL Server](#). You can manage your SQL Virtual Machines with features like automated patching, automated backup, licensing and edition flexibility.

Earlier SQL manageability was offered for only SQL Server Azure marketplace images, but you can now register any Azure virtual machine, with SQL Server installed, with the [SQL VM Resource provider](#) and unlock all manageability features.

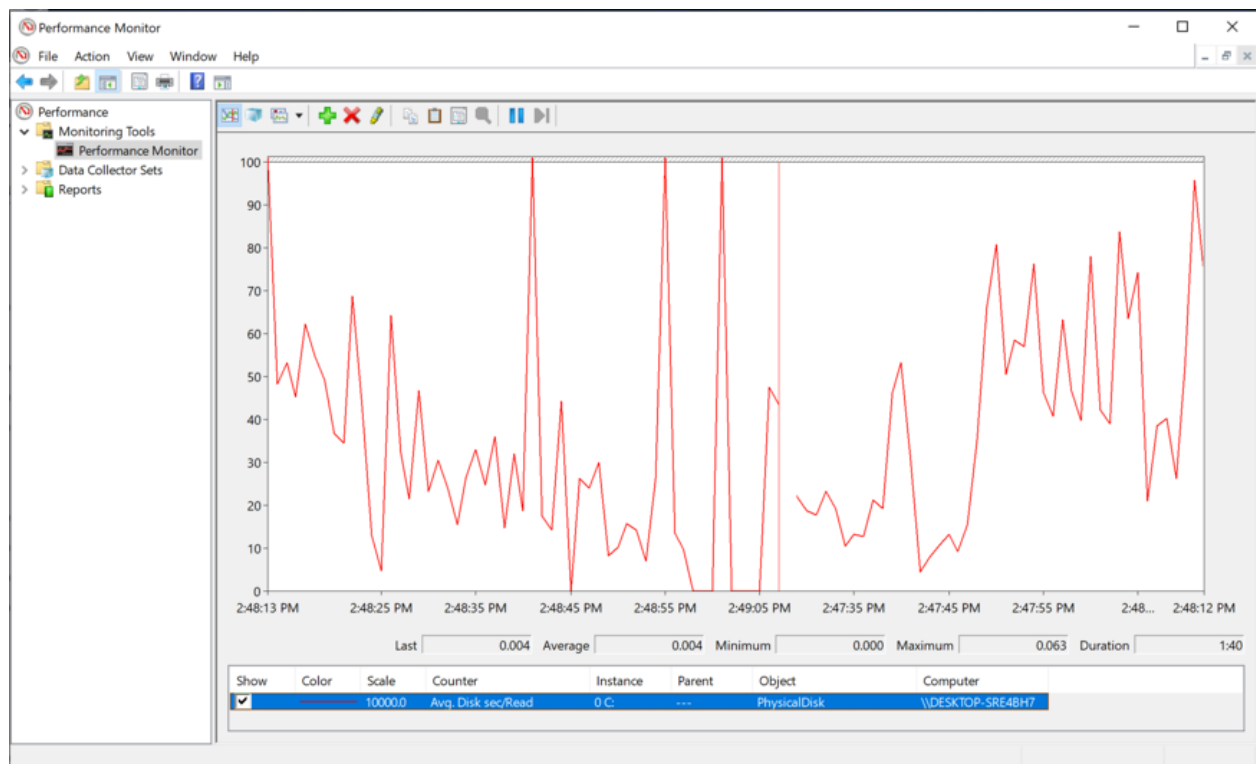
All upcoming manageability features and improvements will only be made available through this new experience.

[Manage SQL virtual machine](#)

This dashboard allows you to see how much space your database files and transaction log file are consuming, and allows you to manage the features provided by the resource provider like automated patching and storage configuration. You can manually install the SQL Resource Provider for other installations of SQL Server on Azure Virtual Machine that weren't defined as part of the virtual machine.

Performance Monitor with SQL Server on an Azure Virtual Machine

Whether you're using an on-premises server or on an Azure Virtual Machine, the Windows Server platform has a native tool called Performance Monitor (commonly shortened to *perfmon* after the name of its executable file) that allows you to easily and routinely monitor performance metrics. *Perfmon* operates with counters for both the operating systems and installed programs. When SQL Server is installed on the operating system, the database engine creates its own group of specific counters.



The image shows the reporting interface of Performance Monitor, with a single counter being collected. This screen is reached from launching Performance Monitor in Windows and shows a live tracker of a specific performance counter. In many cases, you capture multiple counters in the same session. *Perfmon* data can be stored locally and analyzed, but in larger environments you can forward performance monitor results into Azure Monitor, where you can have a single view across many servers.

3. Describe critical performance metrics

<https://learn.microsoft.com/en-us/training/modules/describe-performance-monitoring/3-describe-critical-performance-metrics>

Describe critical performance metrics

Completed

Let's learn how to create metrics in Azure Monitor, which allow you to trigger alerts or execute automated error responses.

Review of Azure metrics

The Azure Monitor service includes the ability to track various metrics about the overall health of a given resource. Metrics are gathered at regular intervals and are the gateway for alerting processes that help to resolve issues quickly and efficiently. Azure Monitor Metrics is a powerful subsystem that allows you to not only analyze and visualize your performance data, but to also trigger alerts that notify administrators or automated actions that can trigger an Azure Automation runbook or a webhook. You can also archive your Azure Metrics data to Azure Storage, since active data is only stored for 93 days.

Create metric alerts

Utilizing the Azure portal, you can create alert rules, based on defined metrics, in the overview section of the Azure Monitor blade. Azure Monitor Alerts can be scoped in three ways. For example, using Azure Virtual Machines as an example you can specify the scope as:

- A list of virtual machines in one Azure region within a subscription
- All virtual machines (in one Azure region) in one or more resource groups in a subscription
- All virtual machines (in one Azure region) in one subscription

In this manner, you can create an alert rule based on resources contained within resource groups as shown.

Microsoft Azure

Search resources, services, and docs (G+I)

Dashboard > Monitor | Alerts

Monitor | Alerts

Search (Cmd+I)

+ New alert rule Manage alert rules Manage actions View classic alerts Refresh

Don't see a subscription? [Open Directory + Subscription settings](#)

Subscription * Dev-Test-Lab Resource group 33 selected Resource

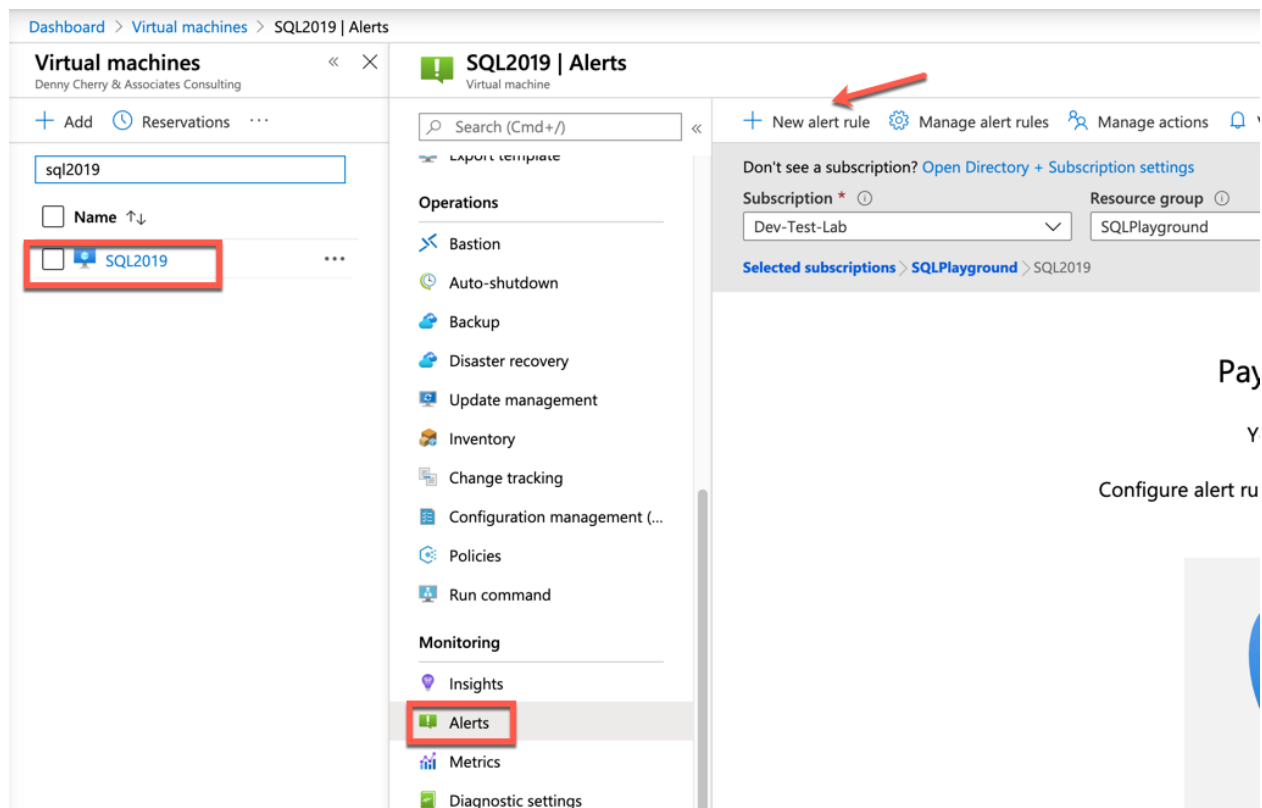
[Selected subscriptions](#) > [Selected resource groups](#)

All is good! You have 1 alert rule

[Manage alert rules \(2\)](#)

The classic alerts can be accessed

The following example demonstrates creating an alert for a virtual machine named *SQL2019*, focusing on the scope of the individual virtual machine.



Regardless the scope of the alert, the creation process is the same.

From the alerts screen, select **New Alert Rule**. If an alert is created from within the scope of a resource, the resource values should be populated for you. You can see that the resource is the *SQL2019* virtual machine, the subscription is *Dev-Test-Lab* and the resource group in which it resides is *SQLPlayground*.

Under the Condition section, select **Add**:

Dashboard > Virtual machines > SQL2019 | Alerts > Create rule

Create rule

Rules management



* RESOURCE

 SQL2019

Select

HIERARCHY

 Dev-Test-Lab >  SQLPlayground



* CONDITION

No condition defined, click on 'Add' to select a signal and define its logic

Add



ACTIONS GROUPS (optional)

Action group name

Contain actions

No action group selected

Add

Create

 Action rules (preview) allows you to define actions at scale as well as suppress actions. Learn more about this functionality by clicking on this banner. 



ALERT DETAILS

Alert rule name * 

Specify alert rule name. Sample: 'Percentage CPU greater than 70'

Description

Specify alert description here...

Select the metric that you wish to alert on. The following image shows Percentage CPU, which you'll then see selected.

9 / 55

Configure signal logic

Choose a signal below and configure the logic on the next screen to define the alert condition.

Signal type ⓘ

All

Monitor service ⓘ

All

Displaying 1 - 20 signals out of total 50 signals

Search by signal name

Signal name	↑↓	Signal type	↑↓	Monitor service	↑↓
Percentage CPU		 Metric		Platform	
Network In Billable (Deprecated)		 Metric		Platform	
Network Out Billable (Deprecated)		 Metric		Platform	
Disk Read Bytes		 Metric		Platform	
Disk Write Bytes		 Metric		Platform	
Disk Read Operations/Sec		 Metric		Platform	
Disk Write Operations/Sec		 Metric		Platform	

The alerts can be configured in a static manner (for example, raise an alert when CPU goes over 95%) or in a dynamic fashion using Dynamic Thresholds. Dynamic Thresholds learn the historical behavior of the metric and raise an alert when the resources are operating in an abnormal manner. These Dynamic Thresholds can detect seasonality in your workloads and adjust the alerting accordingly.

If Static alerts are used, you must provide a threshold for the selected metric. In this example, 80 percent was specified. This threshold means that if the CPU utilization exceeds 80 percentage over a given period, an alert is fired and reacts as specified.

Both types of alerts offer Booleans operators such as the 'greater than' or 'less than' operators. Along with Boolean operators, there are aggregate measurements to select from such as average, minimum, maximum, count, average, and total. With these options available, it's easy to construct a flexible alert that will suit just about any enterprise level alerting.

Configure signal logic

Alert logic

Threshold ⓘ

Static
Dynamic

Operator ⓘ

Greater than

Aggregation type * ⓘ

Average

Threshold value * ⓘ

80

%

Condition preview

Whenever the percentage cpu is greaterthan 80 %

Evaluated based on

Aggregation granularity (Period) * ⓘ

5 minutes

Frequency of evaluation ⓘ

Every 1 Minute

Done

After you create the alert, in order to notify administrators or launch an automation process, an action group needs to be configured.

Defining an action group is optional, and if one isn't configured the alert will just log the notification to storage with no further action. You can create a new action group from the metrics screen, by selecting **Add** next to Action Groups.

Select an action group to attach to this alert rule

The action group selected will attach to this alert rule

+ Create action group

Subscription

Create action group

Contoso Ltd

Once you select **Create Action Group**, you see the following screen. You name the action group and define an alert and the response. In this example, the administrator receives an email notification if

the alert condition is triggered.

Add action group

×

Action group name * ⓘ

CPU Alert for SQL1

✓

Short name * ⓘ

CPUSQL1

✓

Subscription * ⓘ

Contoso Ltd

▼

Resource group * ⓘ

Default-ActivityLogAlerts (to be created)

▼

Actions

Action name *	Action Type *	Status	Configure	Actions
EmailOperator ✓	Email/SMS message/P... ▼		Edit details	×
Unique name for the action	Select an action type ▼			

[Azure Privacy Statement](#)
[Azure Alerts Pricing](#)

ⓘ

Have a consistent format in emails, notifications and other endpoints irrespective of monitoring source. You can enable per action by editing details. Click on the banner to learn more [↗](#)

You can configure the email or SMS details by selecting **Edit Details** under **Configure**, or by adding a new action, which will also bring up the configuration screen.

Email/SMS message/Push/Voice



Add or edit an Email/SMS/Push/Voice action

☒ Email

Email *

☐ SMS (Carrier charges may apply)

Country code

Phone number

☐ Azure app Push Notifications

Azure account email

☐ Voice

Country code

Phone number

Enable the common alert schema. [Learn more](#)

Yes

No

OK

With an action group, there are several ways in which you can respond to the alert. The following options are available for defining the action to take:

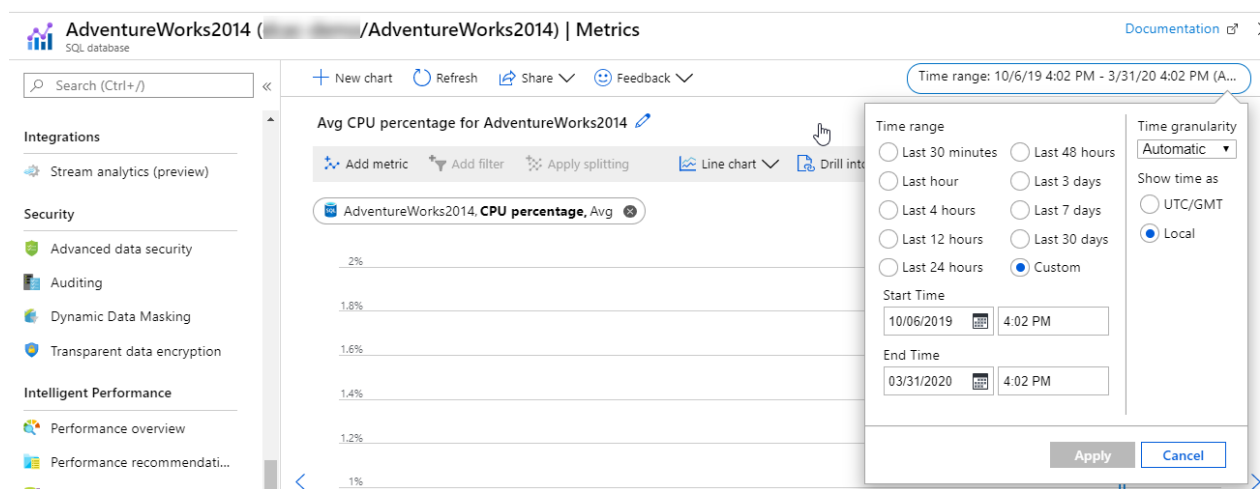
- Automation Runbook
- Azure Function
- Email Azure Resource Manager Role
- Email/SMS/Push/Voice

- ITSM
- Azure Logic App
- Secure Webhook
- Webhook

There are two categories to these actions—notification, which means to notify an administrator or group of administrators of an event, and automation, which is taking a defined action to respond to a performance condition.

Review past performance data

One of the benefits of utilizing the Azure Monitor is the ability to easily and quickly review past metrics that were gathered. If you examine a resource, you note a datetime picker in the upper right-hand corner. Azure Monitor Metrics will be retained for 93 days, after which they're purged, however you do have the option to archive them to Azure Storage.



You're also able to select a smaller window of time such as the last 30 minutes, last hour, last 4 hours, or last 12 hours as an example. The flexibility of Azure monitor allows administrators to quickly identify issues and to potentially diagnose past issues.

SQL Server metrics that matter

Microsoft SQL Server is a well instrumented piece of software that collects a great deal of performance metadata. The database engine has metrics that can be monitored to help identify and improve performance-related issues. Some operating system metrics are only viewable from within performance monitor while others can be accessed through T-SQL queries, in particular, by selecting from the dynamic management views (DMVs). There are some metrics that are exposed in both locations so knowing where to identify specific metrics is important. One example of data that can only be captured from DMVs is data and transaction log file read/write latency as exposed in `sys.dm_os_volume_stats`. On the other hand, an example of an OS metric that isn't available directly through SQL Server is the seconds per disk read and write for the disk volume. Combining

these two metrics can help you gain better understand if a performance issue is related to database structure or a physical storage bottleneck.

4. Establish baseline metrics

<https://learn.microsoft.com/en-us/training/modules/describe-performance-monitoring/4-establish-baseline-metrics>

Establish baseline metrics

Completed

A baseline is a collection of data measurements that helps you understand the normal state of your application or server's performance. Having the data collected over time allows you to identify changes from the normal state. Baselines can be as simple as a chart of CPU utilization over time, or complex aggregations of metrics to offer granular level performance data from specific application calls. The granularity of your baseline depends on the criticality of performance of your database and application.

With any type of application workload, it's imperative to establish a working baseline. A baseline helps you identify if an ongoing issue should be considered within normal parameters or has exceeded given thresholds. Without a baseline, every issue encountered could be considered normal and therefore not require any extra intervention.

Correlating SQL Server and operating system performance

When deploying SQL Server on an Azure virtual machine, it's critical to correlate the performance of SQL Server with the performance of the underlying operating system. If you're using Linux as the operating system, you need to install *InfluxDB*, *Collectd*, and *Grafana* to capture data similar to Windows Performance Monitor. These services collect data from SQL Server and provide a graphical interface to review the data. Utilizing these tools on Linux or Performance Monitor on Windows can be used in conjunction looking at SQL Server-specific data such as SQL Server wait statistics. Using these tools together allow you to identify bottlenecks in hardware or code. The following Performance Monitor counters are a sampling of useful Windows metrics, and can allow you to capture a good baseline for a SQL Server workload:

Processor(_Total)% Processor Time - This counter measures the CPU utilization of all of the processors on the server. It's a good indication of the overall workload, and when used with other counters, this counter can identify problems with query performance.

Paging File(_Total)% Usage - In a properly configured SQL Server, memory shouldn't page to the paging file on disk. However, in some configurations you may have other services running that consume system memory and lead to the operating system paging memory to disk resulting in performance degradation.

PhysicalDisk(_Total)\Avg. Disk sec/Read and Avg. Disk sec/Write - This counter provides a good metric for how the storage subsystem is working. Your latency values in most cases shouldn't be above 20 ms, and with Premium Storage you should see values less than 10 ms.

System\Processor Queue Length - This number indicates the number of threads that are waiting for the time on the processor. If it's greater than zero, it indicates CPU pressure, indicating your workload could benefit from more CPUs.

SQLServer:Buffer Manager\Page life expectancy - Page life expectancy indicates how long SQL Server expects a page to live in memory. There's no proper value for this setting. Older documentation refers to 300 seconds as proper, but that was written in a 32-bit era when servers had far less RAM. You should monitor this value over time, and evaluate sudden drops. Such drops in the counter's value could indicate poor query patterns, external memory pressure (for example, the server running a large SSIS package) or could just be normal system processing like running a consistency check on a large database.

SQLServer:SQL Statistics\Batch Requests/sec - This counter is good for evaluating how consistently busy a SQL Server is over time. Once again there's no good or bad value, but you can use this counter with % Processor time to better understand your workload and baselines.

SQLServer:SQL Statistics\SQL Compilations/sec and SQL Re-Compilations/sec - These counters are updated when SQL Server has to compile or recompile an execution plan for a query because there's no existing plan in the plan cache, or because a plan was invalidated because of a change. Recompiles can indicate T-SQL with recompile query hints, or be indicative of memory pressure on the plan cache caused by either many ad-hoc queries or simple memory pressure.

These counters are just a sample of the available performance monitor counters that are available to you. While these counters provide a good baseline of performance, you may need to examine more counters to identify specific performance problems.

Wait statistics

When a thread is being executed and is forced to wait on an unavailable resource, SQL Server keeps track of these metrics. This information is easily identifiable via the dynamic management view (DMV) `sys.dm_os_wait_stats`. This information is important to understanding the baseline performance of your database, and can help you identify specific performance issues both with query execution and hardware limitations. Identifying the appropriate wait type and corresponding resolution is critical for resolving performance issues. Wait statistics are available across the Azure SQL platform.

5. Explore extended events

<https://learn.microsoft.com/en-us/training/modules/describe-performance-monitoring/5-explore-extended-events>

Explore extended events

Completed

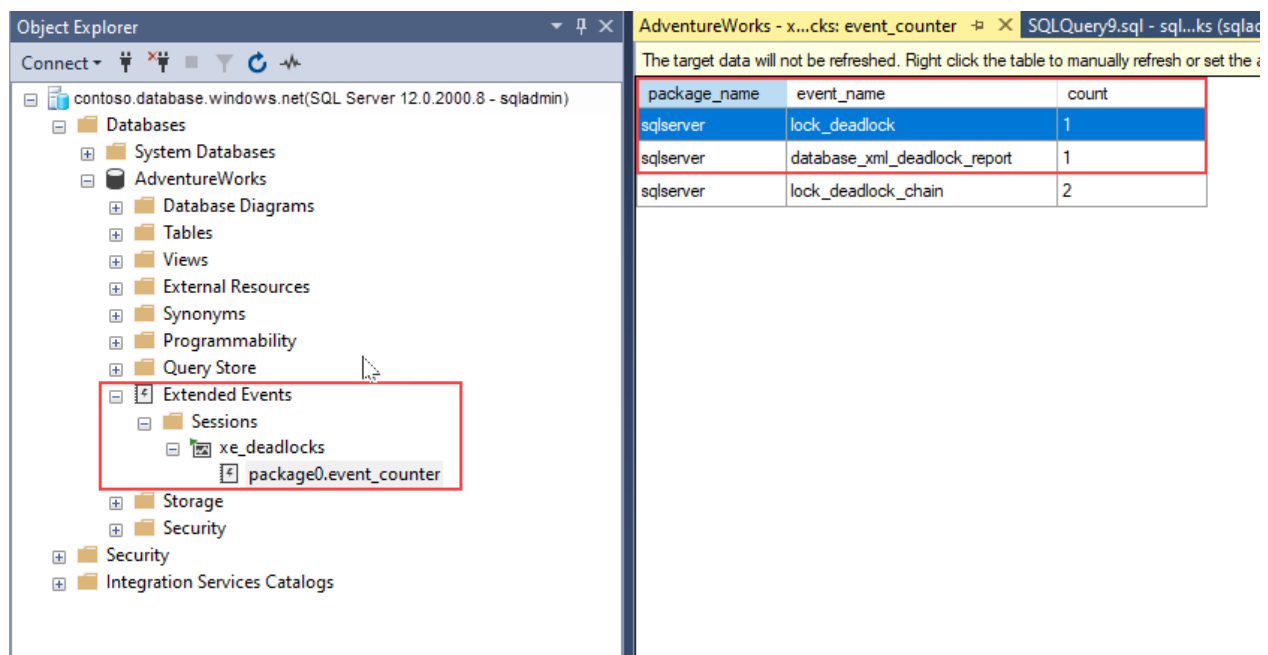
The extended events engine in Azure SQL is a lightweight and powerful monitoring system that allows you to capture granular information about activity in your databases and servers. The monitoring solutions on the Azure platform allow you to easily configure powerful monitoring for your environment and provide automated responses to error conditions.

Extended events build on the functionality of SQL Server Profiler by allowing you to trace queries and by exposing extra data (events) that you can monitor. Some examples of issues you might troubleshoot with Extended Events include:

- Troubleshooting blocking and deadlocking performance issues.
- Identifying long-running queries.
- Monitoring Data Definition Language (DDL) operations.
- Logging missing column statistics.
- Observing Memory Pressure in your database.
- Long-running physical I/O operations.

The extended event framework also allows you to use filters to limit the amount of data you collect in order to reduce the overhead of data collection, and allows you to more easily identify your performance problem by targeting your focus onto specific areas.

The following example shows an extended event session created on Azure SQL Database:



In this example, *xe_deadlocks* is the name an extended event session running on *AdventureWorks* database (on the left side of the image). The *event_counter* target node, which is under your event session node, counts the number of occurrences of each event in the event session. To view the target data in the SSMS Object Explorer, you can select the target node, and then select *View Target Data*. SSMS displays the data as we see on the left side of the image, and the count results for each event.

For more information about extended events on Azure SQL Database, see [Extended events in Azure SQL Database](#).

What can I monitor with extended events?

Extended events cover the full surface area of SQL Server, and are divided into four channels, which define the audience of an event.

- **Admin** - Admin events are targeted for end users and administrators. The events included indicate a problem within a well-defined set of actions an administrator can take. An example of this is the generation of an XML deadlock report to help identity the root cause of the deadlock.
- **Operational** - Operational events are used for analysis and diagnostics or common problems. These events can be used to trigger an action or task based on an occurrence of the event. An example of an operational event would be a database in an availability group changing state, which would indicate a failover.
- **Analytic** - Analytic events are typically related to performance events and are published in high volume. Tracing stored procedure or query execution would be an example of an analytic event.
- **Debug** - Debug events aren't fully documented and you should only use them when troubleshooting with Microsoft support.

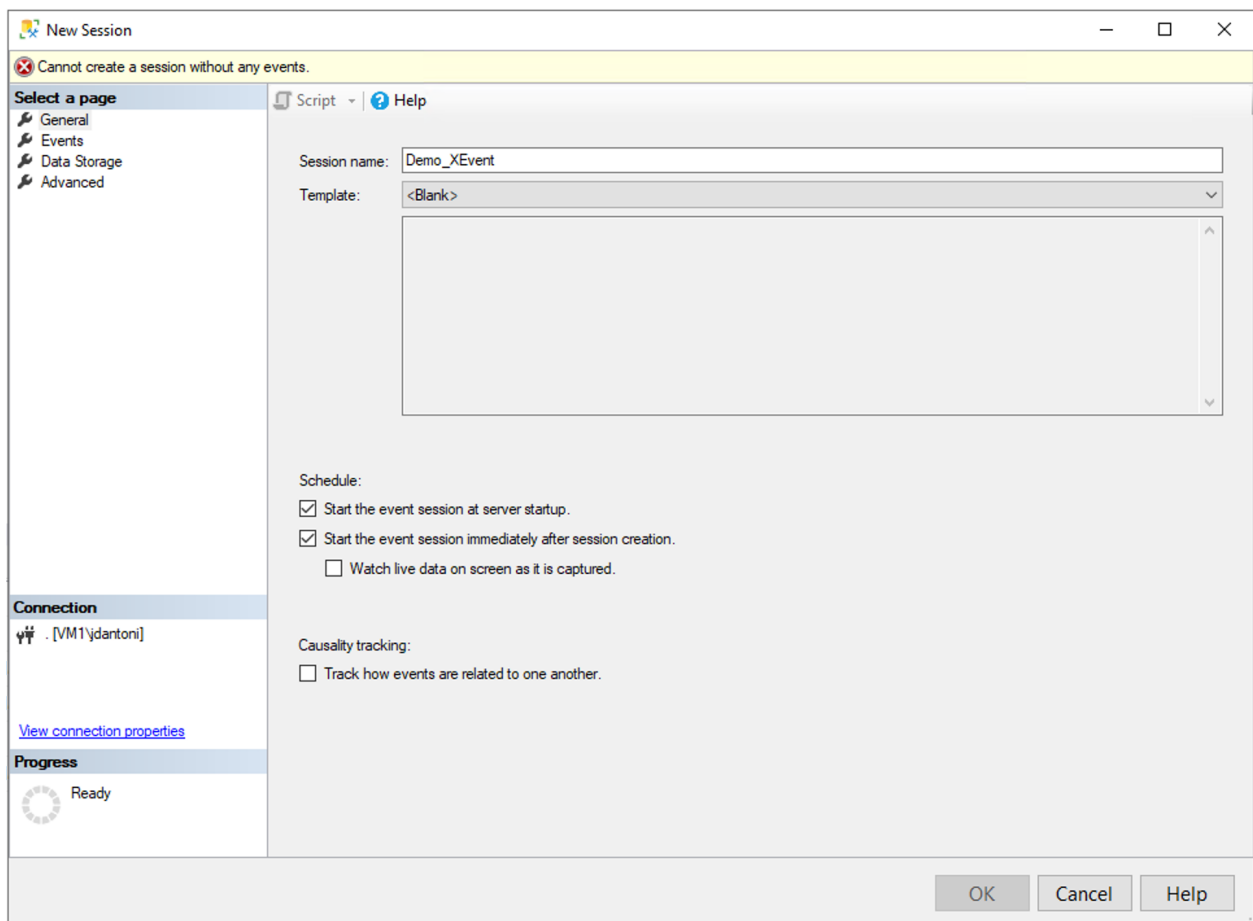
Events are added to sessions, which can host multiple events. Typically, multiple events are grouped together in a session to capture a related set of information.

You can run the following query to obtain a list of the available events, actions, and targets:

```
SELECT
    obj.object_type,
    pkg.name AS [package_name],
    obj.name AS [object_name],
    obj.description AS [description]
FROM sys.dm_xe_objects AS obj
    INNER JOIN sys.dm_xe_packages AS pkg ON pkg.guid = obj.package_guid
WHERE obj.object_type in ('action', 'event', 'target')
ORDER BY obj.object_type,
    pkg.name,
    obj.name;
```

Create extended events session

The following example shows the basic process of creating an extended events session using the *New Session* dialog from SQL Server Management Studio. You can get to this screen by expanding the *Management* node in SSMS, expanding the Extended Events node, select **Sessions**, and then **New Session**.

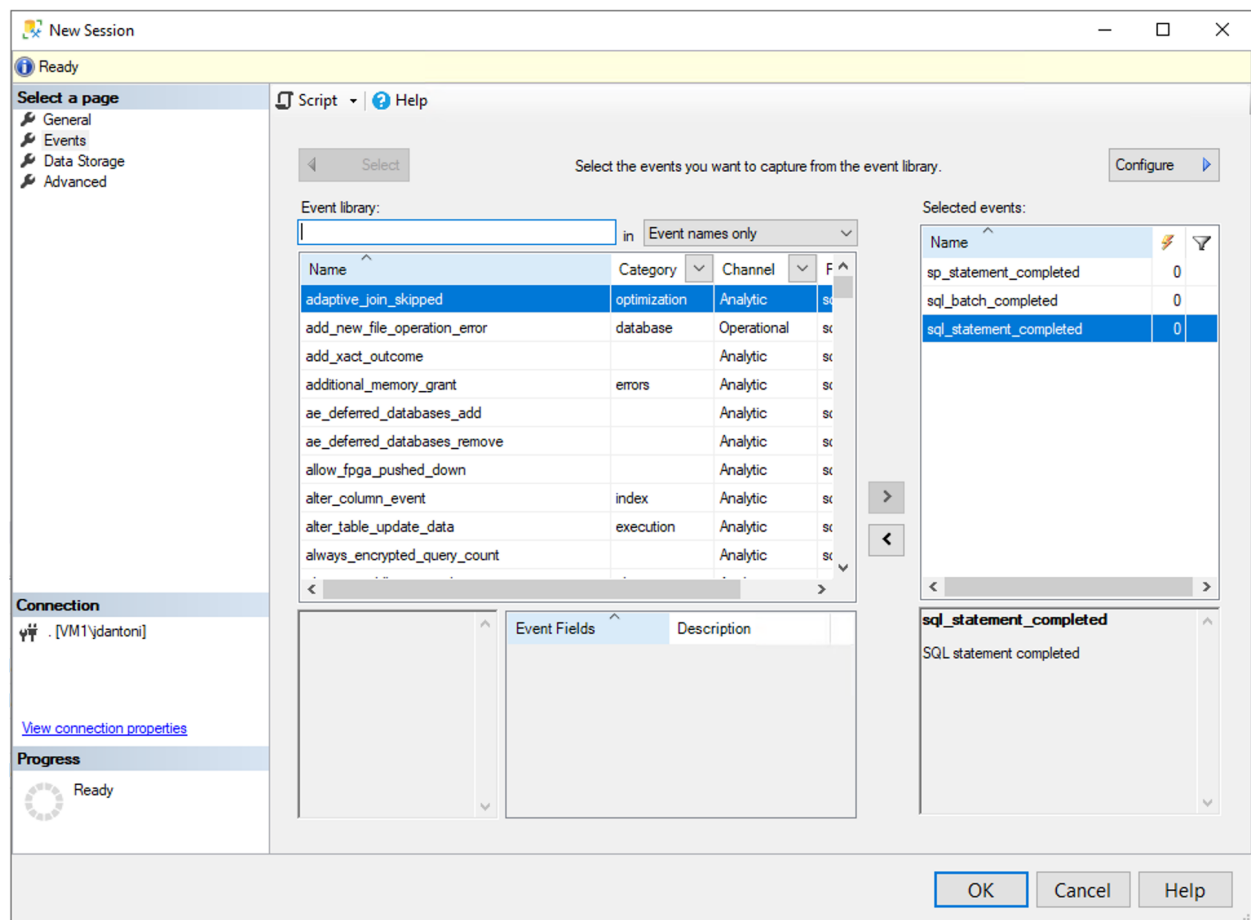


The image shows the *New Session* dialog for the extended events feature. You must first name the session. SQL Server provides numerous templates which are grouped into the following categories:

- Locks and Blocks
- Profiler Equivalents
- Query Execution
- System Monitoring

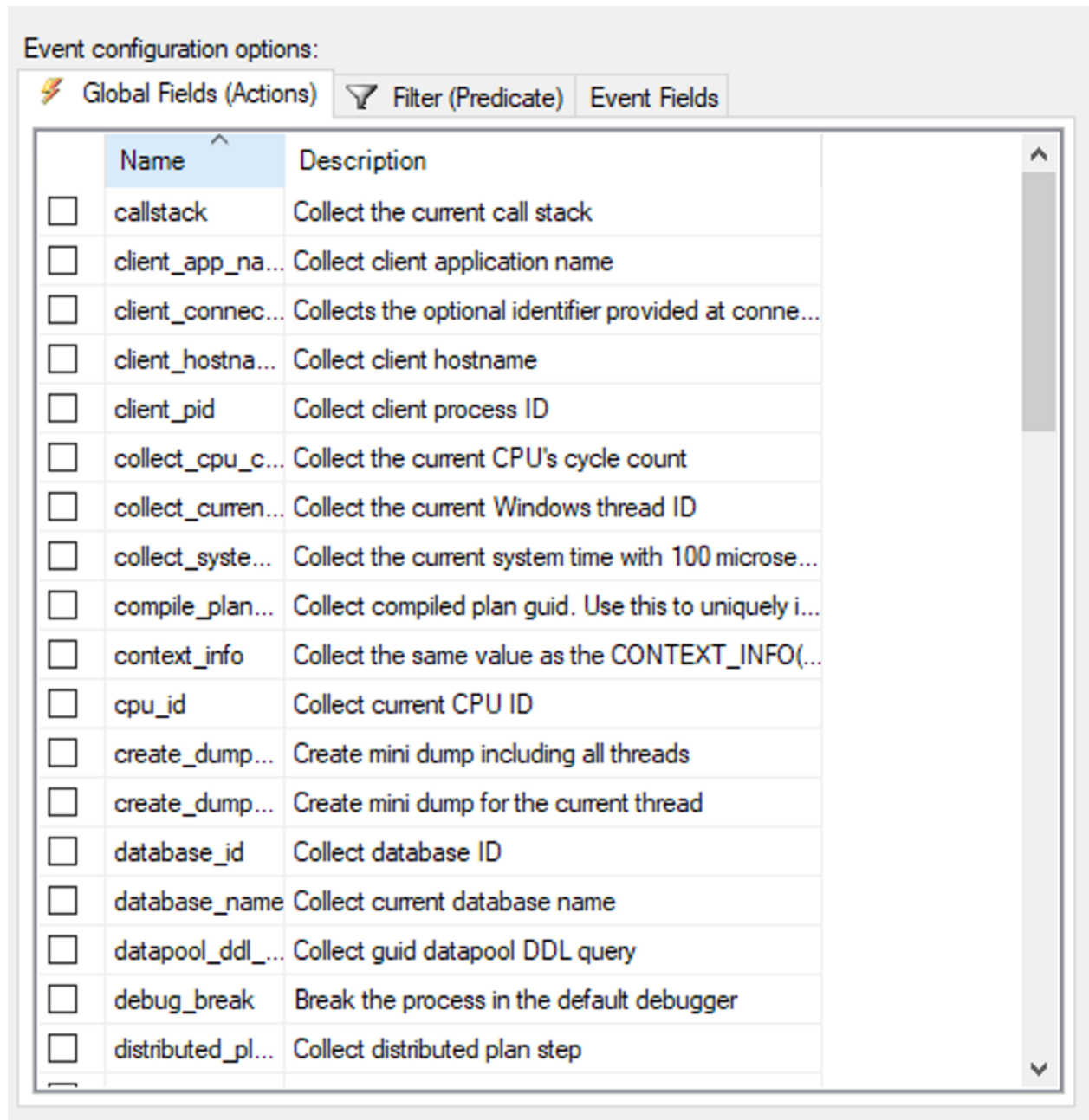
These predefined templates allow you to quickly get started with using extended events for monitoring. In this example, you see events manually added to the session to walk you through all of the options, but when you're getting started, using a template can be an easy way to create a basic session.

You have a couple of check box options for when to start this session. You can choose to have your new session started whenever the server starts, and you can also choose to start the session as soon as it's been created. Administrators can start, and stop extended event sessions at any time through the *Extended Events* node in SQL Server Management Studio. You also have the option of enabling causality tracking, which adds a globally unique identifier (GUID) and sequence number to the output of each event, which allows you to easily step through the order that the events occurred.

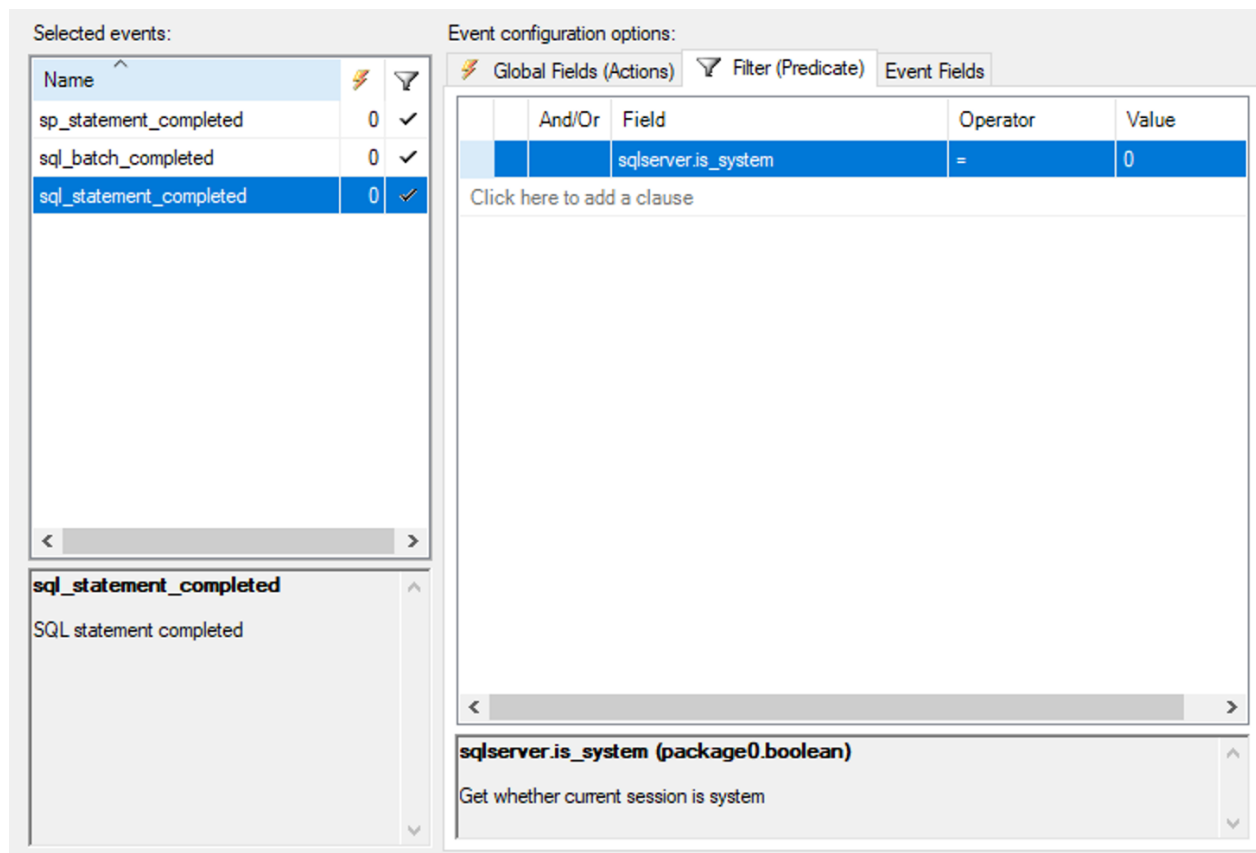


The image shows the screen where you add the events to your session. An event represents a point of interest within the database engine code, and these can represent purely internal system operations, or they can be associated with user actions like query execution. In this example, you can

see that the events `sp_statement_completed`, `sql_batch_completed`, and `sql_statement_completed` have been added to this event session. By default, this session would capture all instances of these events taking place on your instance. You can limit collection by selecting the configure option.



The event configuration screen allows for defining what data you're collecting as it relates to your events. Global fields, allow you to choose the data you're collecting, when your event occurs. Global fields are also known as actions, as the action is to add extra data fields to the event. These fields represent the data that is collected when the extended event occurs, and are common across most extended events. The image shows the filter options for an extended event.



Filters are a powerful feature of Extended Events that allow you to use granular control to capture only the specific occurrences of the event you want to capture. In this example, you can see that filter is being applied on the field `sqlserver.is_system` where it's equal to zero, which indicates that the query isn't an internal operation. In other words, the session won't capture completion of statements that are submitted by system connections, and we only want to capture statements submitted by users or user applications.

Filters apply to a single field on a single event. If you want to make sure that you aren't tracing system activities for any events, you need a separate filter for each: for the `sql_statement_completed` event, for the `sql_batch_completed` event, and for the `sp_statement_completed` event.

It's good practice to configure a filter for each event that you're capturing. This helps improve the efficiency of data collection, and allows you to narrow the focus of your search.

The image shows the event fields that are collected. These are specific to the event being triggered, and can include optional fields for collection. In the above event, you can see the optional collection options are `statement`, and `parameterized_plan_handle`.

Selected events:

Name ^		
sp_statement_completed	0	✓
sql_batch_completed	0	✓
sql_statement_completed	0	✓

sql_statement_completed

SQL statement completed

Event configuration options:

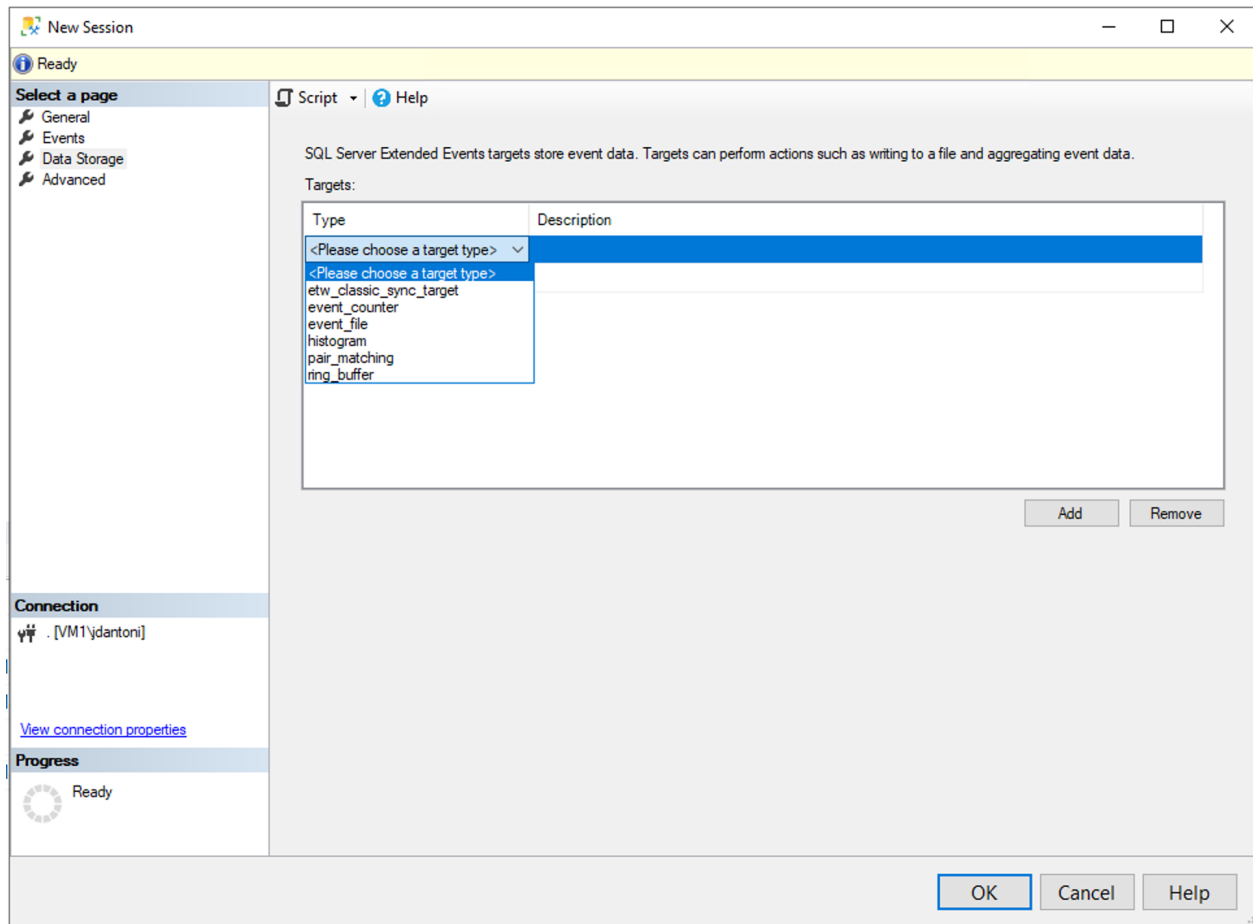
Global Fields (Actions)

Filter (Predicate)

Event Fields

Name ^	Description	Data Type
cpu_time	Indicates the CPU time (in microseconds) ...	uint64
duration	The time (in microseconds) that it took to ...	int64
last_row_count	The last row count for this statement.	uint64
line_number	The statement line number, in relation to t...	int32
logical_reads	The number of logical page reads that we...	uint64
offset	The statement start offset, in relation to th...	int32
offset_end	The statement end offset, in relation to th...	int32
page_server_reads	The number of remote page reads that we...	uint64
☐ parameterized_plan_han...	The plan handle of the cache entry of the...	binary_data
physical_reads	The number of physical page reads that w...	uint64
row_count	The number of rows that were touched by...	uint64
spills	The number of pages that were written w...	uint64
☒ statement	The text of the statement that triggered th...	unicode_string
writes	The number of page writes that were issu...	uint64

Once you have defined an event session, you define a storage target, as shown in the following image.



An extended event session has a target, which is a place for the engine to keep track of occurrences of an event. Two of the more common targets are *event file* which is a file on the file system that can store events, and in Azure SQL PaaS offerings this data is written to a blob storage. Another common target is the *ring buffer* which is within SQL Server's memory. The *ring buffer* is most commonly used for live observation of an event session as it's a circular buffer, and data isn't persisted beyond a session. Most targets process data asynchronously, which means that the event data is written to memory before being persisted to disk. The exception is the Event Tracing for Windows target (ETW), and Event Counter targets, which are processed synchronously.

The following table contains information, and uses for each type of Extended Events target.

Target	Description	Processing
Event Counter	Counts all events that occurred during an Extended Event session. This is used to obtain information about workload characteristics about a workload without the overhead of a full event collection.	Synchronous
Event File	Writes event session output from memory onto persistent file on disk.	Asynchronous
Event Pairing	Many events that generally occur in pairs (for example, lock acquire, lock release), and this collection can be used to identity when those events do no occur in a matched set.	Asynchronous
Event Tracing for Windows (ETW)	Used to correlate SQL Server events with the Windows OS event data.	Synchronous

Target	Description	Processing
Histogram	This is similar to event counter, which counts the occurrences of an event. The difference is that the histogram can count based on a specific event column or action.	Asynchronous
Ring Buffer	Used to hold data in memory. Data isn't persisted to disk and maybe frequently flushed from the buffer	Asynchronous

Alternatively, you can create an extended events session using T-SQL. The following T-SQL commands provide an example of how to create an extended events session:

```
IF EXISTS (SELECT * FROM sys.server_event_sessions WHERE name='test_session')
    DROP EVENT session test_session ON SERVER;
GO

CREATE EVENT SESSION test_session
ON SERVER
    ADD EVENT sqllos.async_io_requested,
    ADD EVENT sqlserver.lock_acquired
    ADD TARGET package0.etw_classic_sync_target (SET default_etw_session_logfile_path
    WITH (MAX_MEMORY=4MB, MAX_EVENT_SIZE=4MB);
GO
```

Event sessions can be scoped to a server or a database. In this example, you're adding two events, and using the Event Tracing for Windows (ETW) path with a file location. After you create the session, you'll have to start it. You can do this through T-SQL, and `ALTER` the session using the `STATE` option, or you can use SQL Server Management Studio for it.

6. Describe database watcher (preview)

<https://learn.microsoft.com/en-us/training/modules/describe-performance-monitoring/6-describe-database-watcher>

Describe database watcher (preview)

Completed

One of the benefits of using any of the products part of the Azure SQL family is the monitoring capability that is built into Azure platform.

Database watcher is a robust monitoring solution designed specifically for [Azure SQL Database](#) and [Azure SQL Managed Instance](#). This tool provides comprehensive insights into the performance, configuration, and overall health of your databases. It collects detailed monitoring data from various

sources, including databases, elastic pools, and SQL managed instances, and ensures that you have a clear and detailed view of your SQL estate.

Note

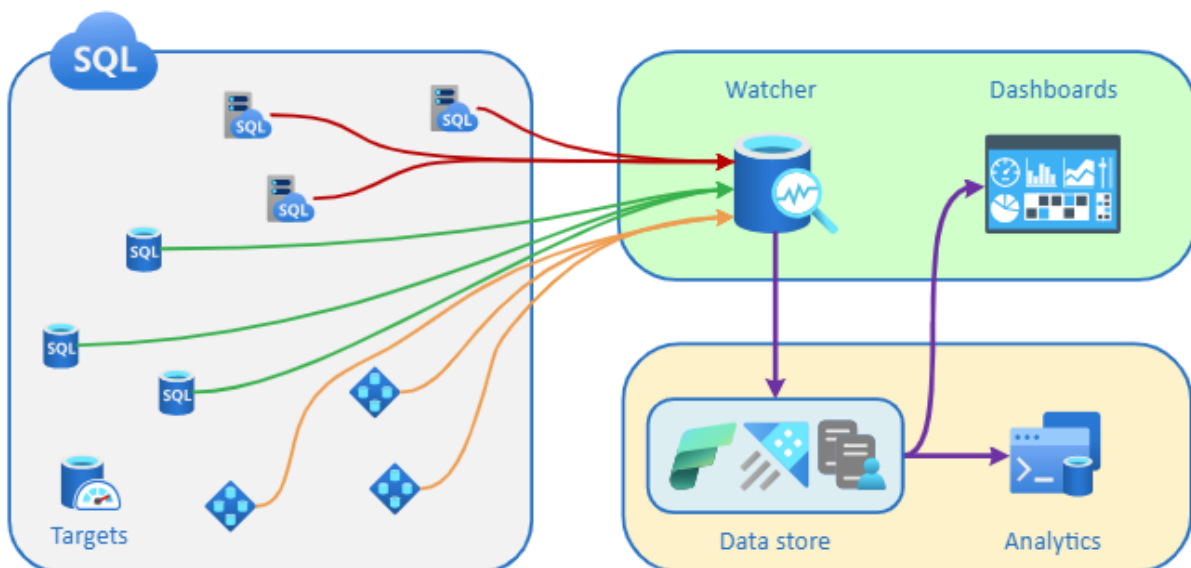
Database watcher is currently in preview.

How database watcher works

One of the key features of Database watcher is its ability to gather and store monitoring data in a central data store within your Azure subscription. This data can then be analyzed using tools like Azure Data Explorer or Real-Time Analytics in Microsoft Fabric, allowing for quick ingestion and analysis of time-series monitoring data.

Database watcher enables you to set targets, datasets, frequency, and retention according to your specific needs. It supports low latency, with data collection and ingestion happening within seconds. This ensures that you always have up-to-date information about your database's performance and health.

This tool also offers a range of dashboards in the Azure portal, providing a single-pane-of-glass view of your entire SQL estate. These dashboards offer detailed insights into each monitored resource, allowing you to quickly identify and address any issues. Additionally, Database Watcher supports parameterized templates for common alert rules, and the ability to create custom alerts, ensuring that you're always notified of critical events.



Database watcher requires a data store for the monitoring data it collects. You can use a database on an [Azure Data Explorer](#) cluster or on a free Azure Data Explorer cluster or you can use a [Real-Time Analytics](#) database in Microsoft Fabric.

Configure database watcher

Here's a simple guide to configure database watcher for Azure SQL Database and Azure SQL Managed Instance:

1. Go to the Azure portal and sign in with your credentials.
2. In the Azure portal, search for "Database Watchers" and select it. Select **Create**.
3. Select your subscription and resource group. Provide a name, and select the region where you want to deploy the database watcher.
4. On the **Data store** tab, configure the data store for the monitoring data.
5. On the **SQL targets** tab, choose the Azure SQL Database or Azure SQL Managed Instance you want to monitor.
6. Review your configuration settings and select **Create** to deploy the database watcher resource.

To learn more about database watcher, see [Quickstart: Create a database watcher to monitor Azure SQL](#).

7. Explore Query Performance Insight

<https://learn.microsoft.com/en-us/training/modules/describe-performance-monitoring/7-explore-query-performance-insight>

Explore Query Performance Insight

Completed

Identifying which queries are consuming the most resources is the first step in any database performance tuning endeavor. In older versions of SQL Server, this required extensive tracing and a series of complex SQL scripts, which could make the process of data gathering cumbersome.

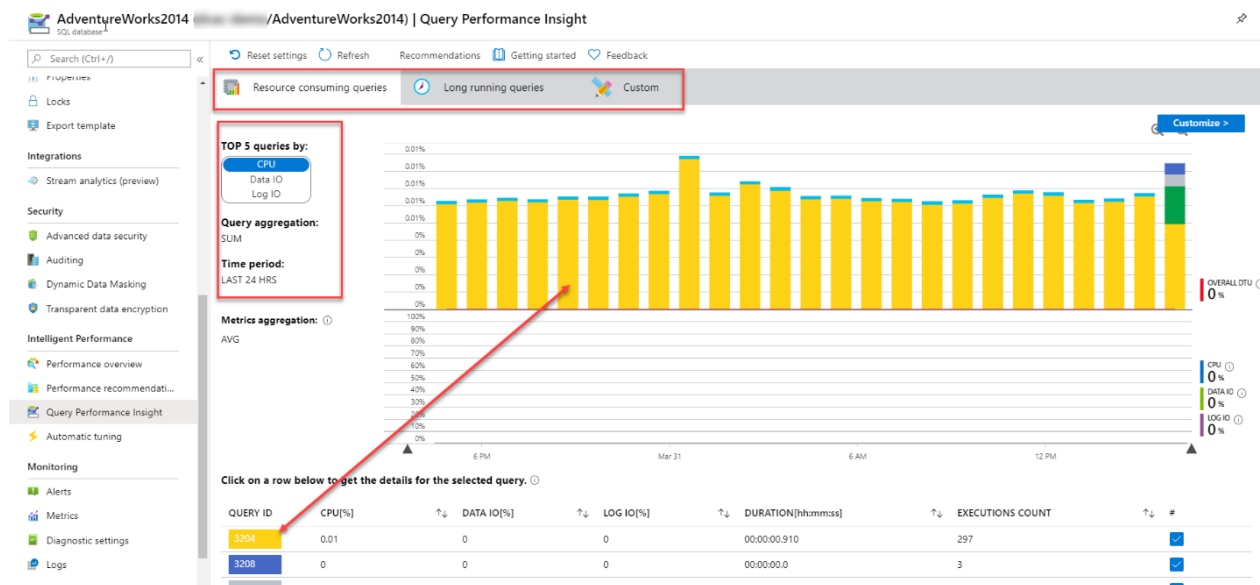
Identify problematic queries

Query Performance Insight allows the administrator to quickly identify expensive queries. You can navigate to Query Performance Insight in the main blade for your Azure SQL Database in the Intelligent Performance section.

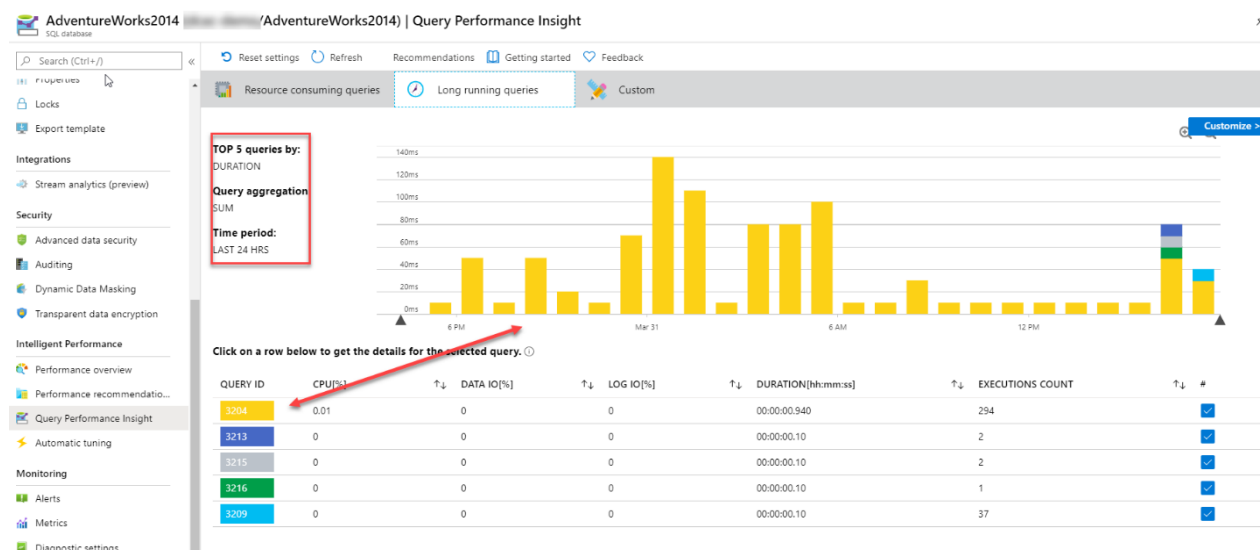
When you launch Query Performance Insight, you discover three buttons to allow you to filter for long running queries, top resource consuming queries, or a custom filter. The default value is

Resource Consuming Queries. This tab shows you the top five queries sorted by the particular resource that you select. In this case, it was sorted by CPU. You also have other options of sorting by Data IO and Log IO metrics.

You can drill into individual queries by selecting the row within the lower grid. Each row is identified with a unique color that correlates to the color within the bar graph above it.

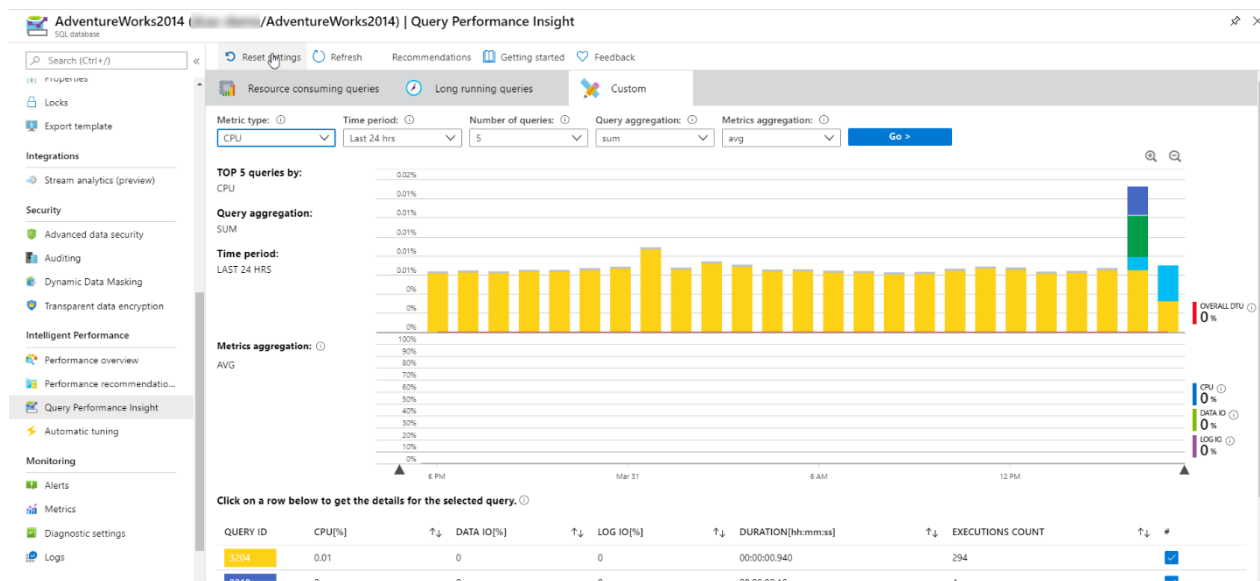


Switching to Long Running Queries, you can see a similar layout as before. In this case, the metrics are limited to the top five queries sorted by duration from the previous 24 hours and is a sum aggregation. In the grid below the graph, you can examine specific queries by selecting the row.



Switching to the custom tab offers more flexibility compared to the other two options.

Within this tab, we can further define how we wish to examine performance data. It offers us several drop-down menus that drive the visual representation of the data. The key metrics are CPU, Log IO, Data IO, and memory. These metrics are the aspects of database performance, the upper limits of which are determined by the service tier and compute resources of your Azure SQL Database.



If we drill into an individual query, we're able to see the query ID and the query itself, as well as the query aggregation type and associated time period. Furthermore, the query ID also correlates to the query ID located within the Query Store. Metrics gleaned from Query Performance Insights can then be easily located within the Query Store itself for deeper analysis or possibly problem resolution if needed.

Query details

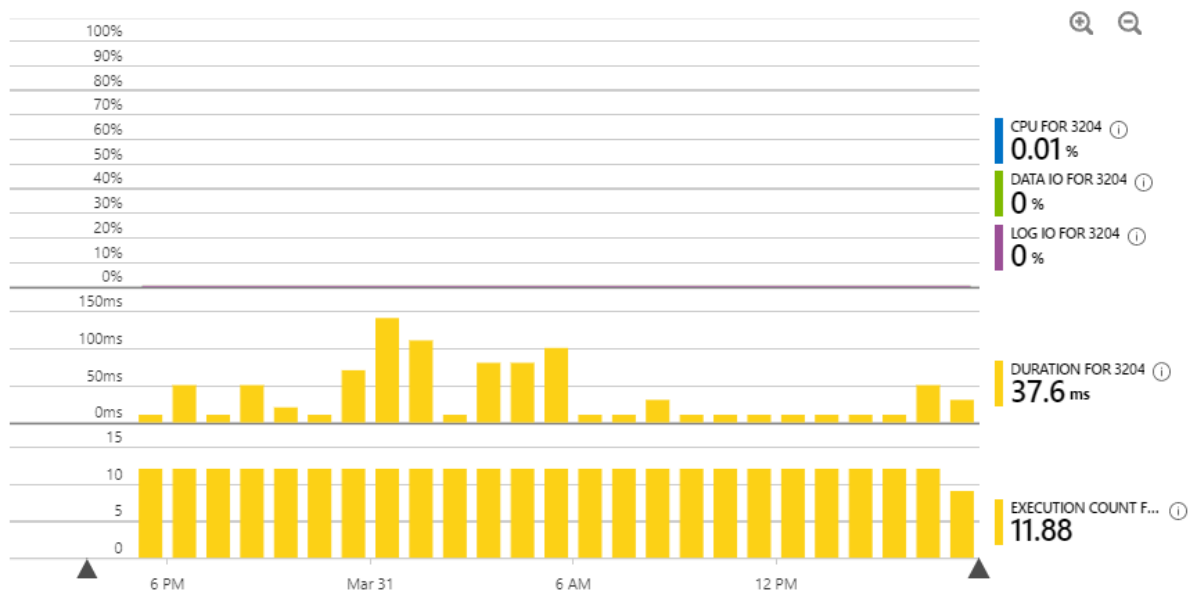
AdventureWorks2014 - Query ID 3204

Settings Refresh Recommendations </> Query Text

Query ID 3204:

```
1 SELECT c.*,
2       i.object_id, i.unique_index_id, i.is_enabled,
3       i.change_tracking_state_desc, i.has_crawl_completed,
4       i.crawl_type_desc, i.crawl_start_date, crawl_end_date,
5       i.incremental_timestamp, i.stoplist_id, i.data_space_id,
6       i.property_list_id,
7       cast(OBJECTPROPERTYEX(i.object_id, 'TableFullTextMergeStatus') as
8       int) as merge_status,
9       cast(OBJECTPROPERTYEX(i.object_id, 'TableFulltextDocsProcessed')
10      as int) as docs_processed,
```

Details of Query ID 3204 (Query aggregation: sum) Last 24 hrs



While Query Performance Insight doesn't show the query's execution plan, you can quickly identify that query, and use the information to extract the plan from the Query Store in your database.

8. Exercise: Isolate problems with monitoring

<https://learn.microsoft.com/en-us/training/modules/describe-performance-monitoring/8-exercise-isolate-problems>

Exercise: Isolate problems with monitoring

Completed

In this exercise, you'll learn how to isolate performance problems through monitoring.

Note

To complete this exercise, you'll need an [Azure subscription](#).

Launch the exercise and follow the instructions.

[Launch Exercise](#)

9. Module assessment

<https://learn.microsoft.com/en-us/training/modules/describe-performance-monitoring/9-knowledge-check>

Module assessment

Completed

10. Summary

<https://learn.microsoft.com/en-us/training/modules/describe-performance-monitoring/10-summary>

Summary

Completed

The Azure platform allows you to use the Azure Monitor to collect baseline performance data about both PaaS and IaaS resources. Within Windows, the Performance Monitor allows you to collect detailed performance information about your SQL Server. Azure SQL Database includes extra detailed query performance information through Query Performance Insight, and provides advanced monitoring capabilities through extended events.

Having a solid understand of the baseline workload of your server and database are critical to understanding performance anomalies.

Configure SQL Server resources for optimal performance

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/configure-sql-server-resources-optimal-performance/1-introduction>

Introduction

Completed

One of the challenges administrators face in the cloud is balancing costs and performance. Both Azure and SQL Server provide many options for configuration to meet the needs of small and large workloads. Choosing the right storage and sizing your virtual machine are critical steps in meeting the performance needs of your applications and balancing cloud costs. Proper configuration of SQL Server resources like TempDB which can easily become a performance bottleneck, and Resource Governor, which can be used to manage multitenant workloads, are also important for properly maintaining your server performance.

Learning objectives

In this module, you will:

- Understand your options for configuration of Azure storage
- Learn how to configure TempDB data files in SQL Server
- Learn how to choose the right type of VM for SQL Server workloads
- Understand the use cases and configuration of Resource Governor in SQL Server

2. Explain how to optimize Azure storage for SQL Server virtual machines

Explain how to optimize Azure storage for SQL Server virtual machines

Completed

Storage performance is a critical component of an I/O heavy application like a database engine. Azure offers a broad array of storage options and can even build your storage solution to meet your workload requirements.

Azure Storage is a robust and secure platform designed to meet the diverse needs of various applications. It offers a wide range of scalable solutions, ensuring that all types of storage support encryption at rest. Users can choose between a Microsoft-managed encryption key or a user-defined encryption key for added security.

- **Blob Storage** - Blob storage is what is known as object-based storage and includes cold, hot, and archive storage tiers. In a SQL Server environment, blob storage will typically be used for database backups, using SQL Server's back up to URL functionality.
- **File Storage** - File storage is effectively a file share that can be mounted inside a virtual machine, without the need to set up any hardware. SQL Server can use File storage as a storage target for a failover cluster instance.
- **Disk Storage** - Azure managed disks offer block storage that is presented to a virtual machine. These disks are managed just like a physical disk in an on-premises server, except that they're virtualized. There are several performance tiers within managed disks depending on your workload. This type of storage is the most commonly used type for SQL Server data and transaction log files.

Azure managed disks

Azure managed disks are block-level storage volumes that are presented to Azure Virtual Machines. Block level storage refers to raw volumes of storage that are created and can be treated as an individual hard drive. These block devices can be managed within the operating system, and the storage tier isn't aware of the contents of the disk. The alternative to block storage is object storage, where files and their metadata are stored on the underlying storage system. Azure Blob Storage is an example of an object storage model. While object storage works well for many modern development solutions, most workloads running in virtual machines use block storage.

The configuration of your managed disks is important to the performance of your SQL Server workloads. If you're moving from an on-premises environment, it's important to capture metrics like

average disk seconds/read and **average disk seconds/write** from Performance Monitor as detailed earlier. Another metric to capture is the I/O Operations per Second, which can be captured using the **SQL Server: Resource Pool Stats Disk Read and Write IO/sec** counters, which show you how many IOPs SQL Server is serving at its peak. It's important to understand your workloads. You want to design your storage and virtual machine to meet the needs of those workload peaks without incurring significant latency. Each Azure Virtual Machine type has a limit on IOPs.

Azure managed disks come in four types:

Ultra disk - Ultra disks support high-IO workloads for mission critical databases with low latency.

Premium SSD - Premium SSD disks are high-throughput and low latency and can meet the needs of most database workloads running in the cloud.

Standard SSD - Standard SSDs are designed for lightly used dev/test workloads or web servers that do a small amount of IO, and require predictable latency.

Standard HDD - Standard HDDs are suitable for backups and file storage that is infrequently accessed.

Typically, production SQL Server workloads use either Ultra disk or Premium SSD, or some combination of the two. Ultra disks are typically used where you're looking for submillisecond latency in response time. Premium SSDs typically have single digit millisecond response time, but have lower costs, and more flexibility in design. Premium SSDs also support read-caching, which can benefit read-heavy database workloads by reducing the number of trips to the disk. The read cache is stored on the local SSD (the D:\ drive on Windows or /dev/sdb1/ on Linux) which can help reduce the number of round trips to the actual disk.

Striping disks for maximum throughput

One of the ways to get more performance and volume out of Azure disks is to stripe your data across multiple disks. This technique doesn't apply to Ultra disk, as you can scale IOPs, throughput, and maximum size independently on a single disk. However, with Premium SSDs it can be beneficial to scale both IOPs and storage volume. In order to stripe disks in Windows, you add the number of disks you would like to the VM, and then create a pool using Storage Spaces in Windows. Don't configure any redundancy for your pool (which would limit your performance) as the redundancy is provided by the Azure framework, which keeps three copies of all disks in synchronous replication to protect against a disk failure. When you create a pool, your pool has the sum of the IOPs and the sum of the volume of all the disks in your pool. For example, if you used 10 P30 disks that are each 1 TB and have 5000 IOPs per disk, you would have a 10-TB volume with 50,000 IOPs available.

SQL Server storage configuration best practices

There are few recommendations for best practices for SQL Server on Azure VMs and their storage configuration:

- Create a separate volume for data and transaction log files
- Enable read caching on the data file volume
- Don't enable any caching on the log file volume
- Plan for an extra 20% of IOPs and throughput when building your storage for your VM to handle workload peaks
- Use the D: drive (the locally attached SSD) for TempDB files because TempDB is recreated upon server restart, so there's no risk of data loss
- Enable instant file initialization to reduce the impact of file-growth activities.
- Move trace file and error log directories to data disks
- For workloads requiring storage latency under 1 millisecond, consider using Ultra disk over Premium SSD.

Azure Virtual Machine resource provider

One way to reduce the complexity of building storage for your SQL Server on an Azure Virtual Machine is to use the SQL Server templates in Azure Marketplace, which allow you to configure your storage as part of your deployment. You can configure the IOPs as needed and the template performs the work of creating your storage spaces pools within Windows.

Configure storage



Storage optimization ⓘ

General

Transactional processing

Data warehousing

Data storage

These disks will be attached to your virtual machine as data disks and will be stored in storage as page blobs.

Data drive location * ⓘ

F:\data



Disk type * ⓘ

Premium SSD



Disk type	Size (GiB)	Max IOPS	Max throughput	Number of disks
1024 GiB, Premium SSD (...)	1024	5000	200	1

1024 GiB, 5000 IOPS, 200 MB/s

Log storage

Transaction logs are a critical component of the database as they record all transactions and database modifications made by each transaction.

Shared drive space * ⓘ

Use a separate drive for lo...

Log drive location * ⓘ

G:\log



Disk type * ⓘ

Premium SSD



Disk type	Size (GiB)	Max IOPS	Max throughput	Number of disks
1024 GiB, Premium SSD (...)	1024	5000	200	1

1024 GiB, 5000 IOPS, 200 MB/s

TempDb storage

The tempDb system database is a global resource that is available to all users connected to the instance of SQL Server. It is used to store temporary user objects and internal objects created by the database engine.

Shared drive space * ⓘ

Use local SSD drive



TempDb drive location * ⓘ

D:\tempDb

This resource provider also supports adding TempDB to the local SSD drive and creates a scheduled task to create the folder on startup.

3. Describe virtual machine resizing

<https://learn.microsoft.com/en-us/training/modules/configure-sql-server-resources-optimal-performance/3-describe-virtual-machine-resizing>

Describe virtual machine resizing

Completed

There are many size options for Azure Virtual Machines. For SQL Server workloads the main characteristics to look for are the amount of memory available, and the number of input and output operations (IOPs) the virtual machine can perform.

Using constrained cores

Typically, SQL Server licenses are based on the number of cores, and Azure allocates a fixed ratio of CPU cores to memory. However, some workloads may require large amounts of memory without needing the default number of allocated CPUs. In such cases, using Azure's constrained cores can be beneficial.

With constrained cores, you can reduce the cost of software licensing while still getting the full amount of memory, storage, and I/O bandwidth. This is good for database workloads that aren't CPU-intensive and can benefit from high memory, storage, and I/O bandwidth, while using a constrained vCPU count.

Using general purpose virtual machines

Most SQL Server production workloads run on the general purpose or memory-optimized families of Azure Virtual Machines. Larger workloads requiring more memory and/or CPU resources land in memory-optimized virtual machines, but many production applications can run comfortably on general purpose virtual machines.

Resizing virtual machines

Azure supports resizing your virtual machine. This operation does require a restart; however, restarting a virtual machine is typically a fast process. In some cases, depending on what virtual machine type you're switching to and from, you may need to deallocate your virtual machine and then resize. This operation does extend the duration of the outage but shouldn't take more than a few minutes.

4. Optimize database storage

<https://learn.microsoft.com/en-us/training/modules/configure-sql-server-resources-optimal-performance/4-optimize-database-storage>

Optimize database storage

Completed

To optimize database storage, you should consider proportional fill and tempdb configuration.

Understand I/O performance

I/O performance can be critical to a database application. Azure SQL abstracts you from physical file placement, but there are methods to ensure you get the I/O performance you need.

Input/output per second (IOPS) might be important to your application. Be sure you've chosen the right service tier and vCores for your IOPS needs. Understand how to measure IOPS for your queries on-premises if you're migrating to Azure. If you have restrictions on IOPS, you might see long I/O waits. In the vCore purchasing model, you can scale up vCores or move to Business Critical or Hyperscale if you don't have enough IOPS. For production workloads, when using DTU, we recommend moving to the Premium tier.

I/O latency is another key component for I/O performance. For faster I/O latency for Azure SQL Database, consider Business Critical or Hyperscale. For faster I/O latency for SQL Managed Instance, move to Business Critical or increase the file size or the number of files for the database. Improving transaction log latency might require you to use multistatement transactions.

Files and filegroups

SQL Server professionals often use files and filegroups to improve I/O performance through physical file placement. Azure SQL doesn't allow users to place files on specific disk systems. However, Azure SQL has resource commitments for I/O performance regarding rates, IOPS, and latencies. In this way, abstracting the user from physical file placement can be a benefit.

Azure SQL Database only has one database file (Hyperscale typically has several), and the maximum size is configured through Azure interfaces. There's no functionality to create more files.

Azure SQL Managed Instance supports adding database files and configuring sizes, but not physical placement of files. You can use the number of files and file sizes for SQL Managed Instance to improve I/O performance. In addition, user-defined filegroups are supported for SQL Managed Instance for manageability purposes.

Describe proportional fill

When inserting 1 gigabyte of data into a SQL Server database with two data files, you might expect each file to increase by approximately 512 megabytes. However, this isn't always the case. SQL Server distributes data based on the size of each file. For instance, if both data files are 2 gigabytes, the data would be evenly distributed. But if one file is 10 gigabytes and the other is 1 gigabyte, around 900 MB would go into the larger file and 100 MB into the smaller one. This behavior is common in any database, but in the write-intensive tempdb, an uneven write pattern can create a bottleneck in the largest file, as it handles more writes.

Configure Tempdb in SQL Server

SQL Server detects the number of available CPUs during setup and configures the appropriate number of files, up to eight, with even sizing. Additionally, the behaviors of trace flags 1117 and 1118 are integrated into the database engine, but only for `tempdb`. For `tempdb`-heavy workloads, it may be beneficial to increase the number of `tempdb` files beyond eight, matching the number of CPUs on your machine.

You use `tempdb` in the same way for both SQL Server and Azure SQL. Note, however, that your ability to configure `tempdb` is different, including the placement of files, the number and size of files, and `tempdb` configuration options.

SQL Server uses `tempdb` for various tasks beyond just storing user-defined temporary tables. It's used for work tables that store intermediate query results, sorting operations, and the version store for row versioning, among other purposes. Due to this extensive utilization, it's crucial to place `tempdb` on the lowest latency storage available and to properly configure its data files.

The database files of `tempdb` are always automatically stored on local SSD drives, so I/O performance shouldn't be an issue.

SQL Server professionals often use more than one database file to partition allocations for `tempdb` tables. For Azure SQL Database, the number of files is scaled with the number of vCores (for example, two vCores equals four files) with a maximum of 16. The number of files isn't configurable through T-SQL against `tempdb`, but you can configure it by changing the deployment option. The maximum size of `tempdb` is scaled per number of vCores. You get 12 files with SQL Managed Instance, independent of vCores.

The database option `MIXED_PAGE_ALLOCATION` is set to `OFF`, and `AUTOGROW_ALL_FILES` is set to `ON`. You can't configure this, but, as with SQL Server, these are the recommended defaults.

The `tempdb` metadata optimization feature introduced in SQL Server 2019, which can alleviate heavy latch contention, isn't currently available in Azure SQL Database or Azure SQL Managed Instance.

Database configuration

Commonly, you configure a database with the T-SQL `ALTER DATABASE` and `ALTER DATABASE SCOPED CONFIGURATION` statements. Many of the configuration options for performance are available for Azure SQL. Consult the [ALTER DATABASE](#) and [ALTER DATABASE SCOPED CONFIGURATION](#) T-SQL reference for the differences between SQL Server, Azure SQL Database, and Azure SQL Managed Instance.

In Azure SQL Database, the default recovery model is full recovery, which ensures that your database can meet Azure service-level agreements (SLAs). This means that minimal logging for bulk operations isn't supported, except for `tempdb`, where minimal logging is allowed.

MAXDOP configuration

Max degree of parallelism (MAXDOP) can affect the performance of individual queries. SQL Server and Azure SQL handle `MAXDOP` in the same way. When `MAXDOP` is set to a higher value, more

parallel threads are used per query, potentially speeding up query execution. However, this increased parallelism requires extra memory resources, which can lead to memory pressure and affect storage performance. For example, when compressing rowgroups into a columnstore, parallelism requires more memory, which can result in memory pressure and rowgroup trimming.

Conversely, setting MAXDOP to a lower value can reduce memory pressure, allowing the storage system to perform more efficiently. This is important in environments with limited memory resources or high storage demands. By carefully configuring MAXDOP, you can balance query performance and storage efficiency, ensuring optimal use of both CPU and storage resources.

You can configure MAXDOP in Azure SQL, similar to SQL Server, by using the following techniques:

- `ALTER DATABASE SCOPED CONFIGURATION` to configure MAXDOP is supported for Azure SQL.
- The stored procedure `sp_configure` for "max degree of parallelism" is supported for SQL Managed Instance.
- MAXDOP query hints are fully supported.
- Configuring MAXDOP with Resource Governor is supported for SQL Managed Instance.

5. Control SQL Server resources

<https://learn.microsoft.com/en-us/training/modules/configure-sql-server-resources-optimal-performance/5-control>

Control SQL Server resources

Completed

While some SQL Servers or Azure SQL managed instances are dedicated to a single application's databases, a configuration often seen in mission-critical applications, many servers support databases for multiple applications with varying performance requirements and peak workload cycles. Balancing these differing requirements can be challenging for administrators. One effective way to manage server resources is by using Resource Governor, introduced in SQL Server 2008.

Resource Governor is a feature in SQL Server and Azure SQL managed instances that allow granular control over CPU, physical I/O, and memory resources for incoming application requests. When enabled at the instance level, Resource Governor uses a classifier function to define how connections are treated, subdividing sessions into workload groups. Each workload group is configured to use a specific pool of system resources.

Resource pools

A resource pool represents the physical resources available on the server. SQL Server always has two pools: default and internal, even when Resource Governor isn't enabled. The internal pool is reserved for critical SQL Server functions and can't be restricted. The default pool, along with any resource pools you explicitly define, can be configured with limits on the resources they can use. For each noninternal pool, you can specify the following limits:

- Min/Max CPU percent
- Cap of CPU percent
- Min/Max memory percent
- NUMA node affinity
- Min/Max IOPs per volume

Note

Changes to a resource pool only impact new sessions, not those already in progress. Therefore, modifying a pool won't restrict the resources of a long-running process. The exception to this rule is external pools used with SQL Server Machine Learning Services, which can be limited by a pool change even for ongoing sessions.

All resource pool settings, except for the minimum and maximum CPU percentage, represent hard limits that can't be exceeded. The min/max CPU percentage only applies when there's CPU contention. For instance, if you set a maximum of 70%, the workload may use up to 100% of available CPU cycles when there's no contention. However, if other workloads are running, the workload will be restricted to 70%.

Workload group

A workload group serves as a container for session requests, classified by the classifier function. Similar to resource pools, there are two built-in groups: default and internal. Each workload group is associated with a single resource pool, but a resource pool can host multiple workload groups. By default, all connections are directed to the default workload group unless the classifier function assigns them to a user-defined group. The default workload group utilizes the resources allocated to the default resource pool.

Classifier function

The classifier function is run at the time a connection is established to the SQL Server instance and classifies each connection into a given workload group. If the function returns a NULL, default, or the name of the nonexistent workload group the session is transferred into the default workload group. Since the classifier is run at every connection, it should be tested for efficiency. The following image shows a sample classifier function that classifies users based on their user name.

```
CREATE FUNCTION dbo.RGClassifier()  
RETURNS SYSNAME
```

```
WITH SCHEMABINDING
AS
BEGIN
DECLARE @WorkloadGroup AS SYSNAME
IF(SUSER_NAME() = 'ReportUser')
    SET @WorkloadGroup = 'ReportServerGroup'

ELSE IF (SUSER_NAME() = 'PrimaryUser')
    SET @WorkloadGroup = 'PrimaryServerGroup'
ELSE
    SET @WorkloadGroup = 'default'

RETURN @WorkloadGroup
END
```

You can increase the complexity of the function definition shown in the example, but you should verify that the more complex function doesn't impact the user performance.

Resource Governor use cases

Resource Governor is used primarily in multitenant scenarios where a group of databases share a single SQL Server instance, and performance needs to be kept consistent for all users of the server. You can also use Resource Governor to limit the resources used by maintenance operations like consistency checks and index rebuilds, to try to guarantee sufficient resources for user queries during your maintenance windows.

6. Module assessment

<https://learn.microsoft.com/en-us/training/modules/configure-sql-server-resources-optimal-performance/6-knowledge-check>

Module assessment

Completed

7. Summary

Summary

Completed

Proper storage configuration is crucial for the performance of your Azure Virtual Machines. To ensure optimal performance, SQL Server should run on premium disk storage or ultra disk. The Resource Provider for SQL Server can automate the creation of storage for your SQL Servers on Azure Virtual Machines, simplifying the setup process. Additionally, Resource Governor can be used to manage to differ workloads within the same SQL Server, allowing you to allocate resources efficiently and maintain balanced performance across various applications. This combination of premium storage and resource management tools ensures that your SQL Server operates smoothly and efficiently in an Azure environment.

Configure databases for optimal performance

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/configure-databases-for-optimal-performance/1-introduction>

Introduction

Completed

In recent versions of SQL Server, Microsoft has moved more configuration options to the database level, giving you more granularity in how your databases behave. Along with those options, they have introduced intelligent query processing features that allow the query optimizer to make better choices.

Even when your database is in the cloud, ongoing performance-related maintenance tasks are critical to the overall success of your applications. Whether it's a SQL Server instance in an Azure Virtual Machine or Azure SQL Database, you need to ensure your statistics are current, and your indexes are well organized.

Learning objectives

In this module, you will:

- Understand database scoped configuration options
- Understand maintenance tasks related to indexing and statistics
- Understand the features of Intelligent Query Processing
- Explore the automatic tuning feature in Azure

2. Explore database maintenance checks

Explore database maintenance checks

Completed

The query optimizer utilizes statistical information from the indexes to attempt to build the most optimal execution plan.

Within Azure SQL maintenance tasks such as backups and integrity checks are handled for you, and while you may be able to get away with automatic updates keeping your statistics up-to-date, sometimes it's not enough.

Having healthy indexes and statistics ensure that any given plan will perform at optimal efficiency. Index maintenance should be performed regularly as data in your databases changes over time. You could change your index maintenance strategy based on the frequency of modifications to your data.

Rebuild and reorganize

Index fragmentation occurs when logical ordering within index pages doesn't match the physical ordering. Pages can get out of order during routine data modification statements such as `UPDATE`, `DELETE`, and `INSERT`. Fragmentation can introduce performance issues because of the extra I/O that is required to locate the data that is being referenced by the pointers within the index pages.

As data is inserted, updated, and deleted from indexes the logical ordering in the index will no longer match the physical ordering inside of the pages, and between the pages, making up the indexes. Also, over time the data modifications can cause the data to become scattered or fragmented in the database. Fragmentation can degrade query performance when the database engine needs to read extra pages in order to locate needed data.

A reorganization of an index is an online operation that will defrag the leaf level of the index (both clustered and nonclustered). This defragmentation process will physically reorder the leaf-level pages to match the logical order of the nodes from left to right. During this process, the index pages are also compacted based on the configured fillfactor value.

A rebuild can be either online or offline depending on the command executed or the edition of SQL Server being utilized. An offline rebuild process will drop and re-create the index itself. If you can do so online, a new index is built in parallel to the existing index. Once the new index has been built, the existing one is dropped and then the new one will be renamed to match the old index name. Keep in mind that the online version requires more space as the new index is built in parallel to the existing index.

The common guidance for index maintenance is:

- **> 5% but < 30%** - Reorganize the index
- **> 30%** - Rebuild the index

Use these numbers as general recommendations. Depending on your workload and data, you may need to be more assertive, or in some cases you may be able to defer index maintenance for databases that mostly perform queries that seek specific pages.

The SQL Server and Azure SQL platforms offer DMVs that allow you to detect fragmentation in your objects. The most commonly used DMVs for this purpose are `sys.dm_db_index_physical_stats` for b-tree indexes, and `sys.dm_db_column_store_row_group_physical_stats` for columnstore indexes.

One other thing to note is that index rebuilds cause the statistics on the index to be updated, which can further help performance. Index reorganization doesn't update statistics.

Microsoft introduced resumable rebuild index operations with SQL Server 2017. Resumable rebuild index operations option provides more flexibility in controlling how much time a rebuild operation might impose on a given instance. With SQL Server 2019, the ability to control an associated maximum degree of parallelism was introduced further providing more granular control to database administrators.

Statistics

When doing performance tuning in Azure SQL, understanding the importance of statistics is critical.

Statistics are stored in the user database as binary large objects (blobs). These blobs contain statistical information about the distribution of data values in one or more columns of a table or indexed view.

Statistics contain information about the distribution of data values within a column. The query optimizer uses column and index statistics in order to determine cardinality, which is the number of rows a query is expected to return.

Cardinality estimates are then used by the query optimizer to generate the execution plan. Cardinality estimates also help the optimizer determine what type of operation (for example, index seek or scan) to use to retrieve the data requested.

To see the list of user defined statistics with the last updated date, run the following query:

```
SELECT sp.stats_id,
       name,
       last_updated,
       rows,
       rows_sampled
FROM sys.stats
     CROSS APPLY sys.dm_db_stats_properties(object_id, stats_id) AS sp
WHERE user_created = 1
```

Create statistics

When you have `AUTO_CREATE_STATISTICS` option to `ON`, the query optimizer creates statistics on the indexed column by default. The query optimizer also creates statistics for single columns in query predicates.

These methods provide high-quality query plans for most queries. At times, you may need to create more statistics using `CREATE STATISTICS` statement to improve specific query plans.

It's recommended to keep the `AUTO_CREATE_STATISTICS` option enabled as it allows the query optimizer to create statistics for query predicate columns automatically.

Whenever you encounter the following situations, consider creating statistics:

- The Database Engine Tuning Advisor suggests creating statistics
- The query predicate contains multiple columns that aren't already in the same index
- The query selects from a subset of data
- The query has missing statistics

Maintenance tasks automation

Azure SQL provides native tools to perform database maintenance tasks for automation purposes. Different tools are available depending on the platform where the database is running.

SQL Server on an Azure Virtual Machine

You have access to scheduling services such as the SQL Agent or the Windows Task Scheduler. These automation tools can help keeping the amount of fragmentation within indexes to a minimum. With larger databases, a balance between a rebuild and a reorganization of indexes must be found to ensure optimal performance. The flexibility provided by SQL Agent or Task Scheduler allows you to run custom jobs.

Azure SQL Database

Due to the nature of Azure SQL Database, you don't have access to SQL Server Agent nor Windows Task Scheduler. Without these services, index maintenance must be created using other methods. There are three ways to manage maintenance operations for SQL Database:

- Azure Automation runbooks
- SQL Agent Job from SQL Server in an Azure Virtual Machine (remote call)
- Azure SQL elastic jobs

Azure SQL Managed Instance

As with SQL Server on an Azure Virtual Machine, you can schedule jobs on a SQL Managed Instance through SQL Server Agent. Using SQL Server Agent provides flexibility to execute code designed to reduce fragmentation within the indexes in the database.

3. Describe database scoped configuration options

<https://learn.microsoft.com/en-us/training/modules/configure-databases-for-optimal-performance/3-describe-database-scoped-configuration-options>

Describe database scoped configuration options

Completed

SQL Server has always offered configuration options at the database level. For example, the recovery model has traditionally been a database setting. As more complex features have been introduced, extra options have been added. Many of these options are linked to the database's compatibility level, which is also a database-level configuration setting. These configuration options can be categorized into two groups, with a minor distinction.

- Options configured by the `ALTER DATABASE SCOPED CONFIGURATION` syntax in T-SQL
- Options configured by the `ALTER DATABASE` syntax in T-SQL

There's no significance to the different ways to set these options. Options that are set using `ALTER DATABASE` include:

- **Database recovery model** – Whether the database is in full or simple recovery model
- **Automatic tuning option** – Whether to enable the force last good plan
- **Auto create and update statistics** – Allows the database to create and update statistics and allows for the option of asynchronous statistics updates
- **Query store options** – The Query Store options are configured here
- **Snapshot isolation** – You can configure snapshot isolation and read committed snapshot isolation

The above settings are a subset of the configurable options.

Many options previously configured on the server can now be configured at the database level. Some of the options include:

- **Maximum Degree of Parallelism** – Allows for a database to configure its own MaxDOP setting and override the server's setting.
- **Legacy Cardinality Estimation** – Allows for the database to use the older cardinality estimator. Some queries may have degraded performance under the newer cardinality estimator, and may benefit from it. You should note that if you use this option with a newer compatibility level, you can still get the benefits of Intelligent Query Processing in compatibility level 140 or 150.
- **Last Query Plan Stats** – Allows you to capture the values of the last actual execution plan for a query. This feature is only active in compatibility level 150.
- **Optimize for Ad Hoc Workloads** – Uses the optimizer to store a stub query plan in the plan cache. This can help reduce the size of the plan cache for workloads that have numerous single use queries.

Database compatibility level

Each database has its own compatibility level, which controls the behavior of the query optimizer for that database.

You can manage this setting when upgrading SQL Server to ensure that your queries have similar execution plans to the older version.

Microsoft supports running on an older compatibility level for an extended period. You should upgrade to a newer compatibility level if possible, as many of the new features in Intelligent Query Processing are only available in compatibility level 140 or 150.

4. Describe automatic tuning

<https://learn.microsoft.com/en-us/training/modules/configure-databases-for-optimal-performance/4-describe-automatic-tuning>

Describe automatic tuning

Completed

Automatic tuning is a monitoring and analysis feature that continuously learns about your workload and identifies potential issues and improvements.

The automatic tuning recommendations are based on the data collected from Query Store. Execution plans evolve over time due to schema changes, index modifications, or changes to the data that

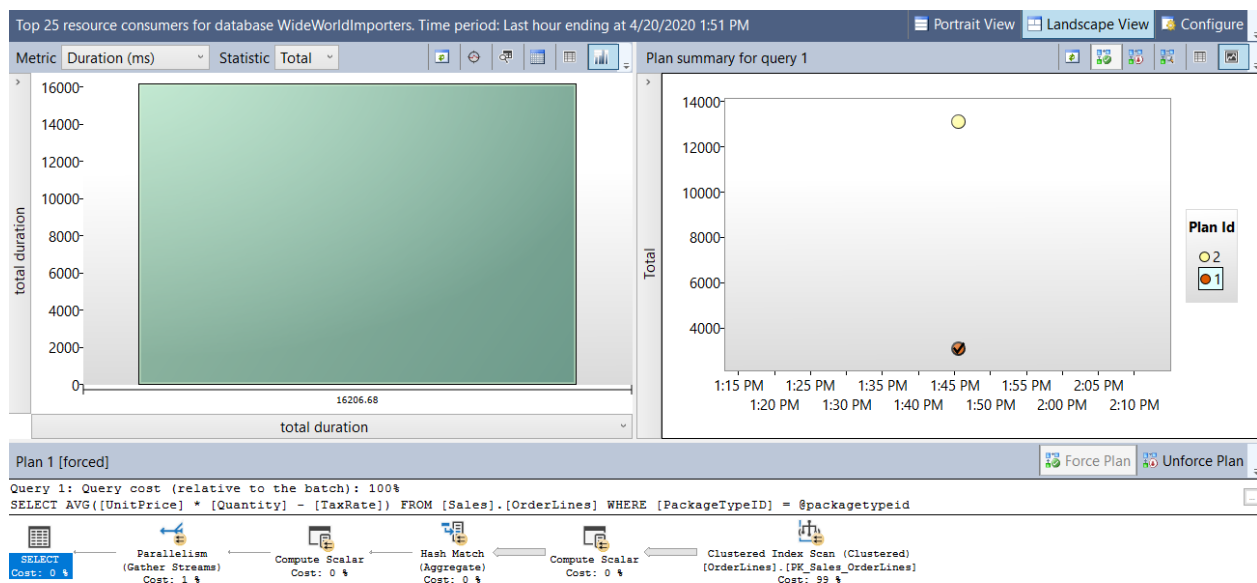
cause updates to the statistics. This evolution can cause queries to perform poorly as the execution plan no longer meets the demands of the given query.

Furthermore, automatic tuning allows for the gathering and applying machine learning services against performance metrics to provide suggested improvements or even allow for self-correction.

Whether on-premises or in the cloud, automatic tuning allows you to identify issues caused by query execution plan regression. Additionally, in Azure SQL Database you can improve query performance by index tuning. Azure SQL Database automatic tuning can identify indexes that should be added or even removed from the database to enhance query performance.

Automatic plan correction

With the help of the Query Store data, the database engine can determine when query execution plans have regressed in performance. While you can manually identify a regressed plan through the user interface, the Query Store also provides the option to notify you automatically.



In the example above, you can see a check mark on **Plan ID 1**, which means that the plan has been forced. After the feature is enabled, the database engine will automatically force any recommended query execution plan, when:

- The previous plan had a higher error rate than the recommended plan
- The estimated CPU gain was greater than 10 seconds
- The force plan has performed better than the previous one

The plan will revert back to the last known good plan after 15 executions of the query.

When plan forcing happens automatically, the database engine applies the last known good plan and keeps an eye on query execution performance. If the forced plan doesn't perform better than the previous one, it's unforced, and a new plan is compiled. However, if the forced plan continues to outperform the previous bad plan, it stays in place until a recompile occurs.

You can enable automatic plan correction via a T-SQL query. The Query Store must be enabled and must be in Read-Write mode for the command to succeed. If either of those two criteria aren't met, the ALTER statement fails.

```
ALTER DATABASE [WideWorldImporters] SET AUTOMATIC_TUNING (FORCE_LAST_GOOD_PLAN = ON)
```

You can examine the automatic tuning recommendations through a dynamic management view (DMV), `sys.dm_db_tuning_recommendations`, which is available in SQL Server 2017 or higher and is also available in Azure SQL Database solutions. This DMV provides information such as reasons as to why the recommendation was provided, the type of recommendation, and the state of the recommendation. To confirm that automatic tuning is enabled for a database, check the view `sys.database_automatic_tuning_options`.

Automatic index management

Azure SQL Database has the capability to perform automatic index tuning. Over time, it learns from existing workloads and provides recommendations for adding or removing indexes to enhance performance. Similar to forcing improved query plans, the database can be configured to automatically create or remove indexes based on their performance, as shown in the following image.

 Azure SQL Database built-in intelligence automatically tunes your databases to optimize performance. Click here to learn more about automatic tuning.

Inherit from: ⓘ

Server Azure defaults Don't inherit

ⓘ The database is inheriting automatic tuning configuration from the server. You can set the configuration to be inherited by going to: [Server tuning settings](#)

Configure the automatic tuning options ⓘ

Option	Desired state	Current state
 FORCE PLAN	ON OFF INHERIT	ON Inherited from server
 CREATE INDEX	ON OFF INHERIT	OFF Inherited from server
 DROP INDEX	ON OFF INHERIT	OFF Inherited from server

When enabled, the **Performance Recommendations** page identifies indexes that can be created or dropped depending on query performance. Remember this feature isn't available for on-premises databases and only available for Azure SQL Database.

Alternatively, use the following query to see the automatic tuning features enabled in your database:

```
SELECT name,
       desired_state_desc,
       actual_state_desc,
       reason_desc
FROM sys.database_automatic_tuning_options
```

Creating new indexes can consume resources, and the timing of the index creations is critical to ensure no negative effect is felt on your workloads.

Azure SQL Database monitors the resources required to implement new indexes to avoid causing performance degradation. The tuning action is postponed until the available resources are available, for example if resources are required for existing workloads and not available for creating an index.

Monitoring ensures any action taken won't harm performance. If an index is dropped and query performance noticeably degrades, the recently dropped index is automatically recreated.

5. Describe intelligent query processing

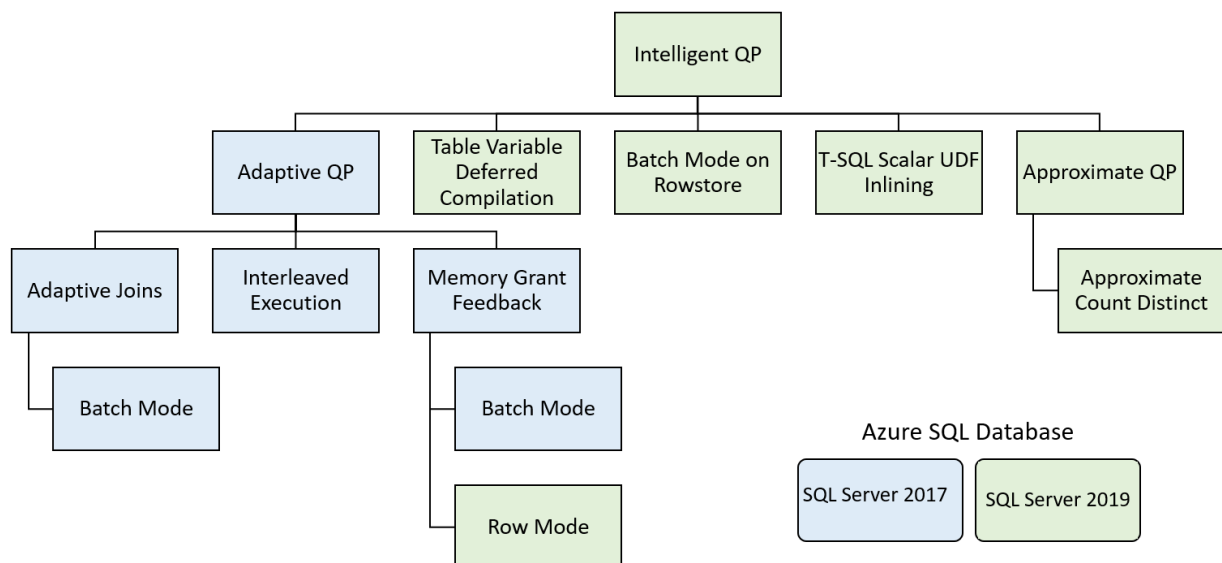
<https://learn.microsoft.com/en-us/training/modules/configure-databases-for-optimal-performance/5-describe-intelligent-query-processing>

Describe intelligent query processing

Completed

In SQL Server 2017 and 2019, and with Azure SQL, Microsoft has introduced many new features into compatibility levels 140 and 150. Many of these features correct what were formerly anti-patterns like using user defined scalar value functions and using table variables.

These features break down into a few families of features:



Intelligent query processing includes features that improve existing workload performance with minimal implementation effort.

To make workloads automatically eligible for intelligent query processing, change the applicable database compatibility level to 150. For example:

```
ALTER DATABASE [WideWorldImportersDW] SET COMPATIBILITY_LEVEL = 150;
```

Adaptive query processing

Adaptive query processing includes many options that make query processing more dynamic, based on the execution context of a query. These options include several features that enhance the processing of queries.

- **Adaptive Joins** – the database engine defers choice of join between hash and nested loops based in the number of rows going into the join. Adaptive joins currently only work in batch execution mode.
- **Interleaved Execution** – Currently this feature supports multi-statement table-valued functions (MSTVF). Prior to SQL Server 2017, MSTVFs used a fixed row estimate of either one or 100 rows, depending on the version SQL Server. This estimate could lead to suboptimal query plans if the function returned many more rows. An actual row count is generated from the MSTVF before the rest of the plan is compiled with interleaved execution.
- **Memory Grant Feedback** – SQL Server generates a memory grant in the initial plan of the query, based on row count estimates from statistics. Severe data skew could lead to either over- or under-estimates of row counts, which can cause over-grants of memory that decrease concurrency, or under-grants, which can cause the query to spill data to tempdb. With Memory Grant Feedback, SQL Server detects these conditions and decreases or increases the amount of memory granted to the query to either avoid the spill or overallocation.

These features are all automatically enabled under compatibility mode 150 and require no other changes to enable.

Table variable deferred compilation

Like MSTVFs, table variables in SQL Server execution plans carry a fixed row count estimate of one row. Much like MSTVFs, this fixed estimate led to poor performance when the variable had a larger row count than expected. With SQL Server 2019, table variables are now analyzed and have an actual row count. Deferred compilation is similar in nature to interleaved execution for MSTVFs, except that it's performed at the first compilation of the query rather than dynamically within the execution plan.

Batch mode on row store

Batch execution mode allows data to be processed in batches instead of row by row. Queries that incur significant CPU costs for calculations and aggregations see the largest benefit from this

processing model. By separating batch processing and columnstore indexes, more workloads can benefit from batch mode processing.

Scalar user-defined function inlining

In older versions of SQL Server, scalar functions performed poorly for several reasons. Scalar functions were executed iteratively, effectively processing one row at a time. They didn't have proper cost estimation in an execution plan, and they didn't allow parallelism in a query plan. With user-defined function inlining, these functions are transformed into scalar subqueries in place of the user-defined function operator in the execution plan. This transformation can lead to significant gains in performance for queries that involve scalar function calls.

Approximate count distinct

A common data warehouse query pattern is to execute a distinct count of orders or users. This query pattern can be expensive against a large table. Approximate count distinct introduces a faster approach to gathering a distinct count by grouping rows. This function guarantees a 2% error rate with a 97% confidence interval.

6. Exercise: Detect and correct fragmentation issues

<https://learn.microsoft.com/en-us/training/modules/configure-databases-for-optimal-performance/6-exercise-detect-correct-fragmentation-issues>

Exercise: Detect and correct fragmentation issues

Completed

In this exercise, you'll learn how to detect and correct fragmentation issues.

Note

To complete this exercise, you'll need an [Azure subscription](#).

Launch the exercise and follow the instructions.

[Launch Exercise](#)

7. Module assessment

<https://learn.microsoft.com/en-us/training/modules/configure-databases-for-optimal-performance/7-knowledge-check>

Module assessment

Completed

8. Summary

<https://learn.microsoft.com/en-us/training/modules/configure-databases-for-optimal-performance/8-summary>

Summary

Completed

Azure SQL has introduced numerous features in recent releases to enhance performance. Many of these features can be enabled at the individual database level and can be managed using the database's compatibility level.

Maintaining indexes and statistics is crucial for ensuring consistent query performance. Azure SQL Database goes a step further by automating the creation of new indexes and the removal of unused ones. This automation helps optimize database performance over time, as the system learns from existing workloads and adjusts indexes accordingly. By leveraging these advanced features, you can achieve better efficiency and reliability in your Azure SQL Database operations.