SQL Server 2022 Features

For Windows, Linux, and Azure

June 2023

Contents

[Overview 7](#_Toc138163644)

[Management 7](#_Toc138163645)

[XML Compression 7](#_Toc138163646)

[CREATE / ALTER TABLE 7](#_Toc138163647)

[CREATE ALTER INDEX​ 8](#_Toc138163648)

[Additional readings 8](#_Toc138163649)

[Shrink database WAIT\_AT\_LOW\_PRIORITY 8](#_Toc138163650)

[Limitations and restrictions 9](#_Toc138163651)

[Recommendations 9](#_Toc138163652)

[Additional readings 10](#_Toc138163653)

[T-SQL Snapshot Backups 10](#_Toc138163654)

[Workflow 10](#_Toc138163655)

[Limitations 13](#_Toc138163656)

[Examples 13](#_Toc138163657)

[Additional reading 16](#_Toc138163658)

[Backup and restore to S3-compatible object storage 16](#_Toc138163659)

[Overview 16](#_Toc138163660)

[Part numbers and file size limitations 16](#_Toc138163661)

[Unsupported features 17](#_Toc138163662)

[Limitations 17](#_Toc138163663)

[Path style and virtual host style 18](#_Toc138163664)

[Examples 18](#_Toc138163665)

[Additional reading 20](#_Toc138163666)

[Accelerated Database Recovery (ADR) 21](#_Toc138163667)

[What is Accelerated Database Recovery (ADR)? 21](#_Toc138163668)

[ADR improvements in SQL Server 2022 (16.x) 21](#_Toc138163669)

[The Current Recovery Process (without ADR) 23](#_Toc138163670)

[Changes with ADR 24](#_Toc138163671)

[New Recovery Process with ADR 25](#_Toc138163672)

[New Overhead to Consider 26](#_Toc138163673)

[Using ADR 26](#_Toc138163674)

[What Workloads can Benefit from ADR? 26](#_Toc138163675)

[Turning ADR On 27](#_Toc138163676)

[Persistent Version Store Size 27](#_Toc138163677)

[Change Location of the PVS to a Different Filegroup 27](#_Toc138163678)

[Database Scoped Configurations 28](#_Toc138163679)

[Database Scoped Configs by Version 28](#_Toc138163680)

[Intelligent Query Processing (IQP) 30](#_Toc138163681)

[Migration Guidance 30](#_Toc138163682)

[Parameter sensitive plan optimization 31](#_Toc138163683)

[Predicate cardinality range 31](#_Toc138163684)

[Degree of Parallelism (DOP) feedback 32](#_Toc138163685)

[Migration Guidance 32](#_Toc138163686)

[Cardinality estimation feedback 32](#_Toc138163687)

[Understand Cardinality Estimation 33](#_Toc138163688)

[CE feedback implementation 33](#_Toc138163689)

[CE feedback scenarios 34](#_Toc138163690)

[Considerations for CE feedback 34](#_Toc138163691)

[Memory Grant Feedback Highlights 35](#_Toc138163692)

[Percentile and persistence mode memory grant feedback 35](#_Toc138163693)

[37](#_Toc138163694)

[Enable memory grant feedback: persistence and percentile 38](#_Toc138163695)

[Disable percentile 38](#_Toc138163696)

[Disable persistence 38](#_Toc138163697)

[Disabling memory grant feedback persistence will also remove existing collected feedback. 38](#_Toc138163698)

[The default setting for MEMORY\_GRANT\_FEEDBACK\_PERSISTENCE is ON. 38](#_Toc138163699)

[Considerations for memory grant feedback 38](#_Toc138163700)

[Migration Guidance 39](#_Toc138163701)

[Optimized plan forcing 39](#_Toc138163702)

[Optimized plan forcing implementation 39](#_Toc138163703)

[Considerations 40](#_Toc138163704)

[42](#_Toc138163705)

[Migration Guidance 42](#_Toc138163706)

[Performance 42](#_Toc138163707)

[Index Updates 42](#_Toc138163708)

[WAIT\_AT\_LOW\_PRIORITY with online index operations 42](#_Toc138163709)

[Ordered clustered columnstore index 43](#_Toc138163710)

[Improved columnstore segment elimination 44](#_Toc138163711)

[Additional Reading 44](#_Toc138163712)

[BufferPool Parallel Scan​ 44](#_Toc138163713)

[Benefits of BufferPool​ Parallel Scan 45](#_Toc138163714)

[Example 46](#_Toc138163715)

[Additional Reading 46](#_Toc138163716)

[TEMPDB 47](#_Toc138163717)

[SQL Server 2022 tempdb improvements 53](#_Toc138163718)

[SQL Server 2022 addresses GAM and SGAM contention 54](#_Toc138163719)

[System page latch concurrency enhancements in SQL Server 2022 55](#_Toc138163720)

[Additional readings 56](#_Toc138163721)

[Query Store on secondary replicas 56](#_Toc138163722)

[Enable Query Store for secondary replicas. 56](#_Toc138163723)

[Additional readings 58](#_Toc138163724)

[Query Store Hints 58](#_Toc138163725)

[When to use Query Store hints 58](#_Toc138163726)

[Query Store hints and feature interoperability 59](#_Toc138163727)

[Query Store hints best practices 60](#_Toc138163728)

[Additional Readings 62](#_Toc138163729)

[Resumable Table Add Constraints 62](#_Toc138163730)

[Online/Resumable Index Operations 62](#_Toc138163731)

[Table and Column Constrains 62](#_Toc138163732)

[Resumable Table Add Constraints 63](#_Toc138163733)

[Resumable Table Add Constraints - Syntax ​ 63](#_Toc138163734)

[Check Resumable Status and Table Constraints 64](#_Toc138163735)

[Additional reading 64](#_Toc138163736)

[Security: 64](#_Toc138163737)

[Always Encrypted with Secure Enclaves 64](#_Toc138163738)

[Protecting Data Through its Lifecycle 64](#_Toc138163739)

[Data at Rest 65](#_Toc138163740)

[Data in transit 65](#_Toc138163741)

[Data in Use 65](#_Toc138163742)

[Always Encrypted 65](#_Toc138163743)

[Always Encrypted with Secure Enclaves 66](#_Toc138163744)

[SQL Ledger 68](#_Toc138163745)

[How it works 70](#_Toc138163746)

[Create an append-only ledger table 72](#_Toc138163747)

[granular permissions 73](#_Toc138163748)

[What is new? 73](#_Toc138163749)

[Extended Events 77](#_Toc138163750)

[Security-related objects 79](#_Toc138163751)

[Backwards-compatibility 79](#_Toc138163752)

[Additional Reading 79](#_Toc138163753)

[Availability 80](#_Toc138163754)

[Contained availability group 80](#_Toc138163755)

[Overview 80](#_Toc138163756)

[Differences 81](#_Toc138163757)

[Contained System Databases 81](#_Toc138163758)

[Connect (Contained environment) 81](#_Toc138163759)

[Read-only routing and contained availability groups 82](#_Toc138163760)

[Differences between connecting to the instance and connecting to the contained availability group 83](#_Toc138163761)

[Additional Reading 83](#_Toc138163762)

[Distributed availability group 83](#_Toc138163763)

[Overview 83](#_Toc138163764)

[Tools 86](#_Toc138163765)

[Azure Data Studio 86](#_Toc138163766)

[Feature comparison with SQL Server Management Studio (SSMS) 86](#_Toc138163767)

[Extensions 87](#_Toc138163768)

[Notebooks 87](#_Toc138163769)

[Multiple Kernels 87](#_Toc138163770)

[Dashboards & Insights Widgets 87](#_Toc138163771)

[SQL Server Management Studio 88](#_Toc138163772)

[SSMS Life Cycle 89](#_Toc138163773)

# Overview

This document provides a general overview of the new features in SQL Server 2022 and Azure SQL. Keep in mind, these features are all inclusive of each version, Widows, Linux, and Azure.

There are many topics in this document. You may find it helpful to open the navigation pane to view the heading of each topic. Home >> Editing >> Find >> Headings.

# Management

## XML Compression

In **SQL Server 2022** XML compression will allow customers to reduce their storage footprint and memory requirements without any driver updates or any other changes needed to your applications. The primary intent of this improvement is to achieve space capacity for the XML data type in SQL Server, though there may be I/O performance improvements for some workloads.​

​

XML compression support will allow customers to reduce their storage footprint without any driver update or changes needed to the application. The primary intent of this improvement is to achieve space savings for the XML data type in **SQL Server 2022**. This improvement targets compression of in-row and off-row XML data in table and compresses the XML indexes.​

​

The sp\_estimate\_data\_compression\_savings supports XML compression estimates and can be used for SQL Server 2022, Azure SQL Database, and Azure SQL Managed Instance.​

EXEC sys.sp\_estimate\_data\_compression\_savings​  
    'Production',   -- schema    ​  
    'ProductModel', -- object   ​  
    NULL,         -- index id   ​  
    NULL,         -- partition #​  
    'PAGE',         -- data compression​  
    1;              -- xml compression

### CREATE / ALTER TABLE

CREATE TABLE Production.ProductModel​  
(​  
    ProductModelId int,​  
    ModelName varchar(50),​  
    ModelDetails xml,​  
)​  
WITH (XML\_COMPRESSION = ON);​  
​  
--WITH​  
--(​  
--    XML\_COMPRESSION = ON ON PARTITIONS (1,2),​  
--    XML\_COMPRESSION = OFF ON PARTITIONS (3)​  
--);​  
​  
ALTER TABLE Production.ProductModel​  
REBUILD PARTITION = ALL ​  
WITH (XML\_COMPRESSION = ON);​

### CREATE ALTER INDEX​

CREATE CLUSTERED INDEX CX\_Production\_ProductModel​

ON Production.ProductModel (ProductModelId)​

WITH (XML\_COMPRESSION = ON);​  
​

CREATE NONCLUSTERED INDEX IX\_ModelName​

ON Production.ProductModel (ModelName)​

WITH (XML\_COMPRESSION = ON);​  
​

ALTER INDEX IX\_ModelName​

ON Production.ProductModel (ModelName)​

REBUILD​

WITH (XML\_COMPRESSION = ON);

### Additional readings

* [CREATE INDEX (Transact-SQL) - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/t-sql/statements/create-index-transact-sql?view=sql-server-ver16)

## Shrink database WAIT\_AT\_LOW\_PRIORITY

In previous releases, shrinking databases and database files to reclaim space often leads to concurrency issues. SQL Server 2022 (16.x) adds WAIT\_AT\_LOW\_PRIORITY as an additional option for shrink operations (DBCC SHRINKDATABASE and DBCC SHRINKFILE). When you specify WAIT\_AT\_LOW\_PRIORITY, new queries requiring Sch-S or Sch-M locks aren't blocked by the waiting shrink operation, until the shrink operation stops waiting and begins executing. See [Shrink a database](https://learn.microsoft.com/en-us/sql/relational-databases/databases/shrink-a-database?view=sql-server-ver16) and [Shrink a file](https://learn.microsoft.com/en-us/sql/relational-databases/databases/shrink-a-file?view=sql-server-ver16).

Shrink operations in progress can block other queries on the database and can be blocked by queries already in progress. Introduced in SQL Server 2022 (16.x), shrink database operations have a WAIT\_AT\_LOW\_PRIORITY option. This feature is an additional new option for DBCC SHRINKDATABASE and DBCC SHRINKFILE. If a new shrink operation in WAIT\_AT\_LOW\_PRIORITY mode can't obtain the necessary locks due to a long-running query already in progress, the shrink operation will eventually time out after one minute and silently exit, preventing other queries from being blocked.

### Limitations and restrictions

* The database can't be made smaller than the minimum size of the database. The minimum size is the size specified when the database was originally created, or the last explicit size set by using a file-size-changing operation, such as DBCC SHRINKFILE. For example, if a database was originally created with a size of 10 MB and grew to 100 MB, the smallest size the database could be reduced to is 10 MB, even if all the data in the database has been deleted.
* You can't shrink a database while the database is being backed up. Conversely, you can't back up a database while a shrink operation on the database is in process.

### Recommendations

* To view the current amount of free (unallocated) space in the database. For more information, see [Display Data and Log Space Information for a Database](https://learn.microsoft.com/en-us/sql/relational-databases/databases/display-data-and-log-space-information-for-a-database?view=sql-server-ver16)
* Consider the following information when you plan to shrink a database:
* A shrink operation is most effective after an operation that creates a large amount of unused storage space, such as a large DELETE statement, truncate table, or a drop table operation.
  + Most databases require some free space to be available for regular day-to-day operations. If you shrink a database repeatedly and notice that the database size grows again, this indicates that the free space is required for regular operations. In these cases, repeatedly shrinking the database is a wasteful operation. Auto grow events necessary to grow the database file(s) hinder performance.
  + A shrink operation doesn't preserve the fragmentation state of indexes in the database, and generally increases fragmentation to a degree. This is another reason not to shrink the database repeatedly.
  + Unless you have a specific requirement, don't set the AUTO\_SHRINK database option to ON.

### Additional readings

* [Shrink a database - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/relational-databases/databases/shrink-a-database?view=sql-server-ver16)

## T-SQL Snapshot Backups

Databases are getting larger and larger every day. Traditionally, SQL Server backups are streaming backups. A streaming backup depends on the size of the database. Backup operations consume resources (CPU, memory, I/O, network) which affect throughput of the concurrent OLTP workload for the duration of the backup. One way to make the backup performance constant, rather than depend on the size of data, is by performing a snapshot backup using mechanisms provided by the underlying storage hardware or service.

Because the backup itself happens at the hardware level, this isn't a pure SQL Server solution. SQL Server must first prepare the data and log files for the snapshot, so that the files are guaranteed to be in a state that can later be restored. Once this is done, I/O is frozen on SQL Server, and control is handed over to the backup application to complete the snapshot. Once the snapshot has successfully completed, the application must return control back to SQL Server where I/O is then resumed. Because we must freeze I/O for the duration of the snapshot operation, it is essential that the snapshot happens quickly, so that the workload on the server isn't interrupted for an extended period. In the past, users have relied on third-party solutions that were built on top of the SQL Writer service to complete snapshot backups. The SQL Writer service depends on Windows VSS (Volume Shadow Service) along with SQL Server VDI (Virtual Device Interface) to perform the orchestration between SQL Server and the disk-level snapshot. Backup clients based on the SQL Writer service tend to be complex, and they only work on Windows. With T-SQL snapshot backups, the SQL Server side of the orchestration can be handled with a series of T-SQL commands. This allows users to create their own small backup applications that can run on either Windows or Linux, or even scripted solutions if the underlying storage supports a scripting interface to initiate a snapshot.

Here is a sample PowerShell script that demonstrates an end-to-end solution of backing up and restoring a database in an Azure SQL IaaS Virtual Machine using the T-SQL snapshot backup capabilities introduced in SQL Server 2022 (16.x) (and higher).

### Workflow

The T-SQL snapshot backup syntax decouples the vendor-dependent snapshot mechanism from the suspend and backup operations. With this syntax, you can:

Freeze a database with ALTER command – providing an opportunity for you to perform the snapshot of the underlying storage. After which, you can thaw the database and record the snapshot with BACKUP command.

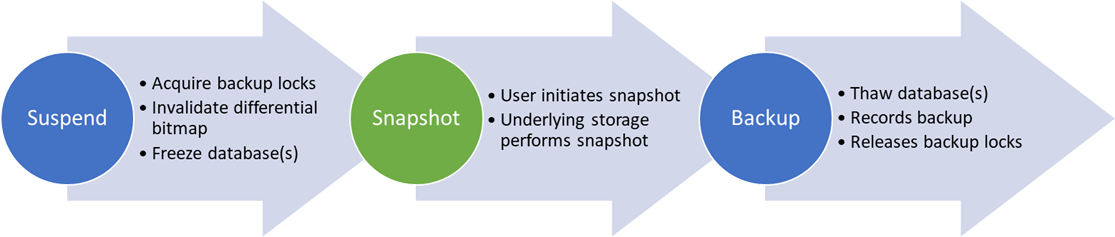
Perform snapshots of multiple databases simultaneously with the new BACKUP GROUP and BACKUP SERVER commands. With this option, snapshots can be performed at the snapshot granularity of the underlying storage, eliminating the need to perform a snapshot of the same disk multiple times.

Perform FULL backups and COPY\_ONLY FULL backups. These backups are recorded in msdb as well.

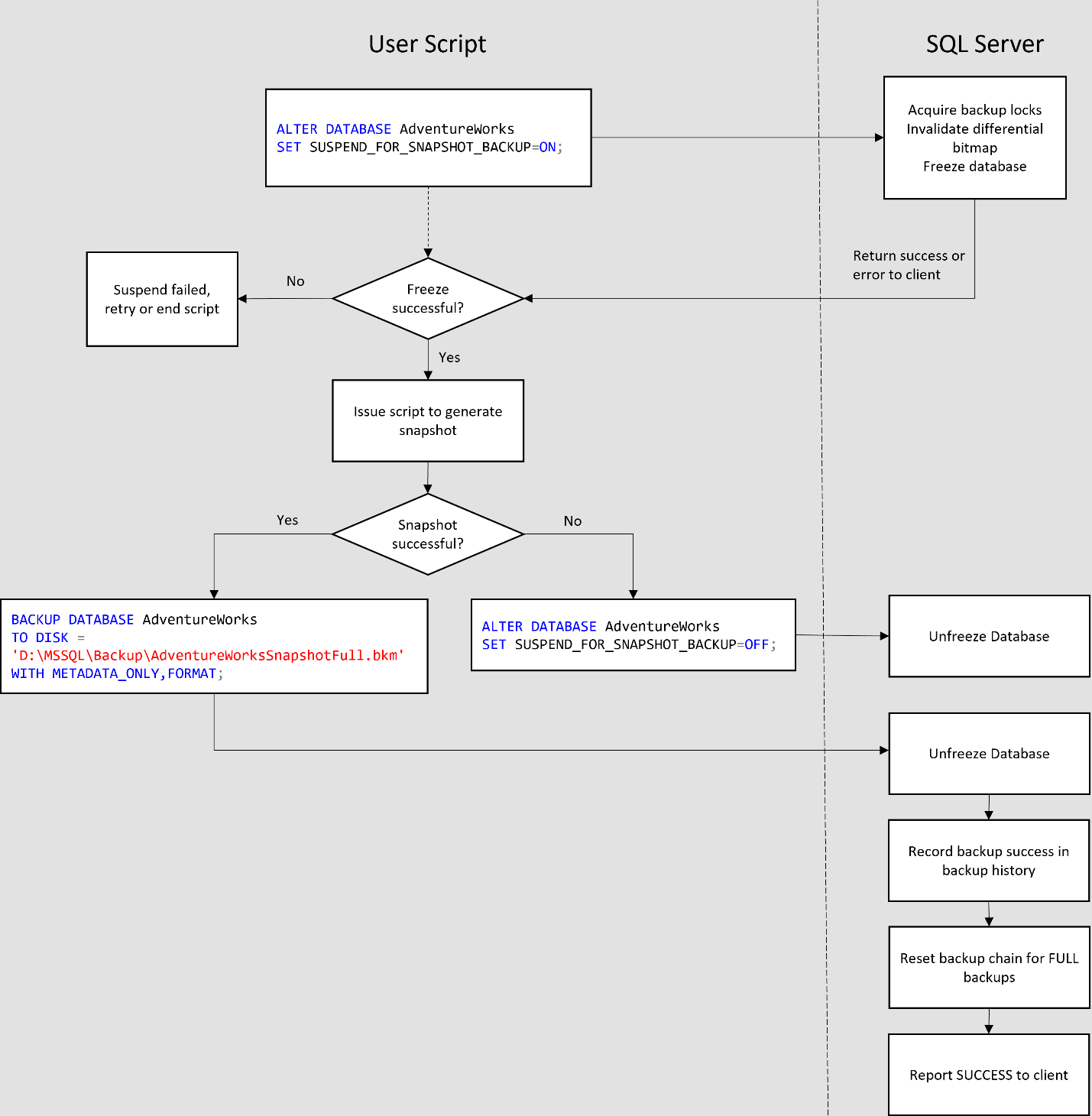
Perform point-in-time recovery using log backups taken with the normal streaming approach after the snapshot FULL backup. Streaming differential backups are also supported if desired.

This new feature now makes it possible for users to create their own small backup solutions that can run on either Windows or Linux. ​

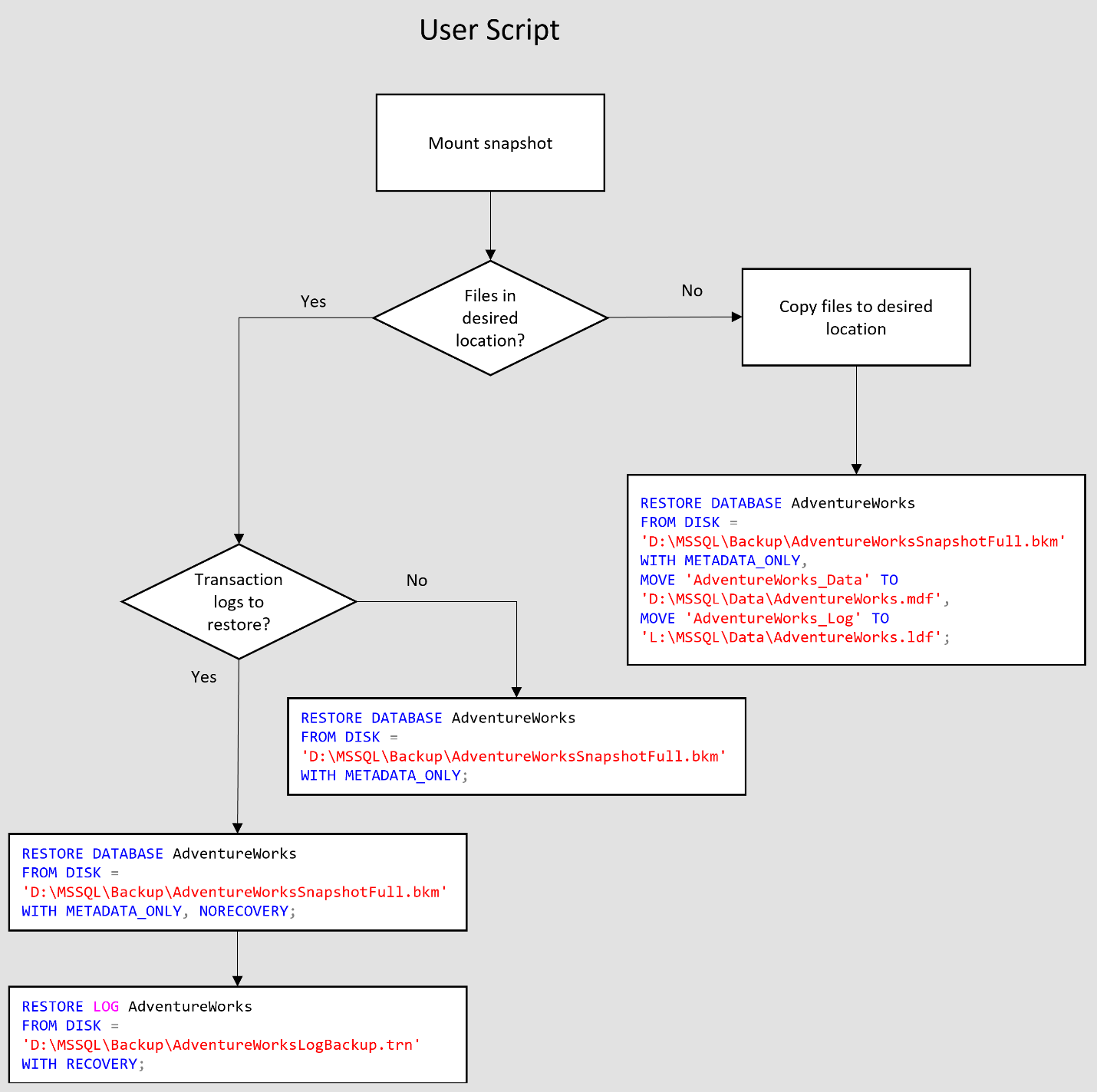
The following diagram illustrates the high-level workflow of T-SQL snapshot backups:

[](https://learn.microsoft.com/en-us/sql/relational-databases/backup-restore/media/create-a-transact-sql-snapshot-backup/t-sql-snapshot-backup-workflow.png?view=sql-server-ver16#lightbox)

The middle snapshot step requires you to initiate the snapshot on the underlying storage. The following diagram shows an example of how a backup script might work with SQL Server to complete the snapshot backup process:

[](https://learn.microsoft.com/en-us/sql/relational-databases/backup-restore/media/create-a-transact-sql-snapshot-backup/backup.png?view=sql-server-ver16#lightbox)

Similarly, a restore script might work as follows:

[](https://learn.microsoft.com/en-us/sql/relational-databases/backup-restore/media/create-a-transact-sql-snapshot-backup/restore.png?view=sql-server-ver16#lightbox)

### Limitations

The maximum number of databases you can back up with this feature is 64. If there are more than 64 databases on the server, you will see the following error:

Error message:

Msg 925, Level 19, State 1, Line 4

Maximum number of databases used for each query has been exceeded. The maximum allowed is 64.

### Examples

The following sections show different T-SQL commands used to perform snapshot backup to disk. When a snapshot backup is written on the disk, only the metadata connected to the snapshot backup is written to the file. The output doesn't contain any of the database contents except for the header and the file contents. The shell file created as part of performing snapshot backup should be used with the actual snapshot URI to make a complete backup. RESTORE of a database from this file requires the user to copy the database files from the snapshot URI to the mount point prior to issuing the RESTORE command. Users can run all the traditional T-SQL commands like RESTORE HEADERONLY, RESTORE FILELISTONLY on this snapshot backup metadata file along with RESTORE DATABASE. The syntax supports writing snapshot backup metadata to DISK or URL. The snapshot backup sets can also be applied just like streaming backup sets into a single file.

**Suspend a single user database for snapshot backup and record a database backup.**

ALTER DATABASE testdb1

SET SUSPEND\_FOR\_SNAPSHOT\_BACKUP = ON;

BACKUP DATABASE testdb1

TO DISK = 'd:\temp\db.bkm'

WITH METADATA\_ONLY, FORMAT;

**Suspend multiple user databases for snapshot backup.**

If multiple databases on the same underlying disk, you could suspend multiple databases with the following command.

ALTER SERVER CONFIGURATION

SET SUSPEND\_FOR\_SNAPSHOT\_BACKUP = ON

(GROUP = (testdb1, testdb2));

BACKUP GROUP testdb1, testdb2

TO DISK = 'd:\temp\db.bkm'

WITH METADATA\_ONLY, FORMAT;

**Suspend all user databases on the server for snapshot backup.**

If all the user databases on the server need to be suspended, use the following command.

ALTER SERVER CONFIGURATION

SET SUSPEND\_FOR\_SNAPSHOT\_BACKUP = ON;

BACKUP SERVER

TO DISK = 'd:\temp\db.bkm'

WITH METADATA\_ONLY, FORMAT;

**Suspend multiple user databases with a single command.**

Record snapshot of all the user databases on the server into a single backup set:

ALTER SERVER CONFIGURATION

SET SUSPEND\_FOR\_SNAPSHOT\_BACKUP = ON

(GROUP = (testdb1, testdb2));

BACKUP GROUP testdb1, testdb2

TO DISK = 'd:\temp\db.bkm'

WITH METADATA\_ONLY, FORMAT;

**Perform copy-only snapshot backup.**

Since the differential bitmap is cleared prior to freeze, SUSPEND\_FOR\_SNAPSHOT\_BACKUP provides an option (COPY\_ONLY) to not clear the differential bitmap prior to freeze.

ALTER DATABASE testdb1

SET SUSPEND\_FOR\_SNAPSHOT\_BACKUP = ON

(MODE = COPY\_ONLY);

BACKUP DATABASE testdb1

TO DISK = 'd:\temp\db.bkm'

WITH METADATA\_ONLY, FORMAT;

ALTER SERVER CONFIGURATION

SET SUSPEND\_FOR\_SNAPSHOT\_BACKUP = ON

(GROUP = (testdb1, testdb2), MODE = COPY\_ONLY);

BACKUP GROUP testdb1, testdb2

TO DISK = 'd:\temp\db.bkm'

WITH METADATA\_ONLY, FORMAT;

ALTER SERVER CONFIGURATION

SET SUSPEND\_FOR\_SNAPSHOT\_BACKUP = ON

(MODE = COPY\_ONLY);

BACKUP SERVER

TO DISK = 'd:\temp\db.bkm'

WITH METADATA\_ONLY, FORMAT;

### Additional reading

* [Create a Transact-SQL snapshot backup - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/relational-databases/backup-restore/create-a-transact-sql-snapshot-backup?view=sql-server-ver16)

## Backup and restore S3-compatible object storage.

### Overview

SQL Server 2022 (16.x) introduces object storage integration to the data platform, enabling you to integrate SQL Server with S3-compatible object storage in addition to Azure Storage. To provide this integration the SQL Server has been enhanced with a new S3 connector, which uses the S3 REST API to connect to any provider of S3-compatible object storage. SQL Server 2022 (16.x) extends the existing BACKUP/RESTORE TO/FROM URL syntax by adding support for the new S3 connector using the REST API.

URLs pointing to S3-compatible resources are prefixed with s3:// to denote that the S3 connector is being used. URLs beginning with s3:// always assume that the underlying protocol is https.

### Part numbers and file size limitations

To store data, the S3-compatible object storage provider must split files into multiple blocks called parts. This is similar to [block blobs](https://learn.microsoft.com/en-us/rest/api/storageservices/understanding-block-blobs--append-blobs--and-page-blobs) in Azure Blob Storage.

Each file can be split up to 10,000 parts, each part size ranges from 5 MB to 20 MB, this range is controlled by the T-SQL BACKUP command through the parameter [MAXTRANSFERSIZE](https://learn.microsoft.com/en-us/sql/t-sql/statements/backup-transact-sql?view=sql-server-ver16#with-options). The default value of MAXTRANSFERSIZE is 10 MB, therefore the default size of each part is 10 MB.

The maximum supported size of a single file is the result of 10,000 parts \* *MAXTRANSFERSIZE*, if it is required to backup a bigger file it must split/striped up to 64 URLs. The final maximum supported size of a file is 10,000 parts \* *MAXTRANSFERSIZE* \* URLs.

##### Region

Your S3-compatible object storage provider can offer the ability to determine a specific region for the bucket location. The use of this optional parameter can provide more flexibility by specifying which region that bucket belongs to. This parameter requires the use of WITH together with either BACKUP\_OPTIONS or RESTORE\_OPTIONS. These options require the value to be declared in JSON format.

If no value is declared, us-east-1 is assigned as default.

**Backup example:**

WITH

BACKUP\_OPTIONS = '{"s3": {"region":"us-west-1"}}'

**Restore example:**

WITH

RESTORE\_OPTIONS = '{"s3": {"region":"us-west-1"}}'

##### Linux support

SQL Server uses Win HTTP to implement the client of HTTP REST APIs it uses. It relies on OS certificate store for validations of the TLS certificates being presented by HTTP(s) endpoint. However, SQL Server on Linux the CA must be placed on a predefined location to be created at /var/opt/mssql/security/ca-certificates, only the first 50 certificates can be stored and supported in this folder.

SQL Server reads the certificates from the folder during startup and adds them to the trust store.

Only super users should be able to write in the folder, while the mssql user must be able to read.

### Unsupported features

* Backup to S3-compatible object storage with a nonsecure HTTP URL is not supported. Customers are responsible for setting up their S3 host with an HTTPS URL and this endpoint is validated by a certificate installed on the SQL Server OS host.
* Backup to S3-compatible object storage is not supported in SQL Server Express and SQL Server Express with Advanced Services editions.

### Limitations

The following are the current limitations of backup and restore with S3-compatible object storage:

1. Due to the current limitation of S3 Standard REST API, the temporary uncommitted data files that are created in the customer's S3-compatible object store (due to an ongoing multipart upload operation) while the BACKUP T-SQL command is running, are not removed in case of failures. These uncommitted data blocks continue to persist in S3-compatible object storage in the case the BACKUP T-SQL command fails or is canceled. If the backup succeeds, these temporary files are automatically removed by the object store to form the final backup file. Some S3-compatible storage providers handle this through their garbage collector system.
2. The total URL length is limited to 259 characters. The full string is counted in this limitation, including the s3:// connector name. So, the usable limit is 254 characters. However, we recommend sticking to a limit of 200 characters to allow for possible introduction of query parameters.
3. The SQL credential name is limited to 128 characters in UTF-16 format.
4. Secret key ID only supports alphanumeric values.

### Path style and virtual host style

Backup to S3 supports the URL to be written in both path style and virtual host style.

Path style example: s3://<endpoint>:<port>/<bucket>/<backup\_file\_name>

Virtual host example: s3://<bucket>.<domain>/<backup\_file\_name>

### Examples

##### Create credentials.

* The IDENTITY should always be 'S3 Access Key' when using the S3 connector.
* The Access Key ID and Secret Key ID must not contain a colon. Access Key ID and Secret Key ID are the user and password created on the S3-compatible object storage.
* Only alphanumeric values are allowed.
* The Access Key ID must have proper permission on the S3-compatible object storage.

The following examples create SQL Server credentials for authentication with the object storage endpoint:

CREATE CREDENTIAL [s3://<endpoint>:<port>/<bucket>]

WITH

IDENTITY = 'S3 Access Key',

SECRET = '<AccessKeyID>:<SecretKeyID>';

##### Backup to URL

The following example performs a full database backup to the object storage endpoint, striped across multiple files:

BACKUP DATABASE <db\_name>

TO URL = 's3://<endpoint>:<port>/<bucket>/<database>\_01.bak'

, URL = 's3://<endpoint>:<port>/<bucket>/<database>\_02.bak'

, URL = 's3://<endpoint>:<port>/<bucket>/<database>\_03.bak'

--

, URL = 's3://<endpoint>:<port>/<bucket>/<database>\_64.bak'

WITH FORMAT -- overwrite

, STATS = 10

, COMPRESSION.

##### Restore from URL

The following example performs a database restore from the object storage endpoint location:

RESTORE DATABASE <db\_name>

FROM URL = 's3://<endpoint>:<port>/<bucket>/<database>\_01.bak'

, URL = 's3://<endpoint>:<port>/<bucket>/<database>\_02.bak'

, URL = 's3://<endpoint>:<port>/<bucket>/<database>\_03.bak'

--

, URL = 's3://<endpoint>:<port>/<bucket>/<database>\_64.bak'

WITH REPLACE -- overwrite

, STATS = 10;

##### Options for encryption and compression

The following example shows how to back up and restore the AdventureWorks2019 database with encryption, MAXTRANSFERSIZE as 20 MB and compression:

CREATE MASTER KEY ENCRYPTION BY PASSWORD = <password>;

GO

CREATE CERTIFICATE AdventureWorks2019Cert

WITH SUBJECT = 'AdventureWorks2019 Backup Certificate';

GO

-- Backup database

BACKUP DATABASE AdventureWorks2019

TO URL = 's3://<endpoint>:<port>/<bucket>/AdventureWorks2019\_Encrypt.bak'

WITH FORMAT, MAXTRANSFERSIZE = 20971520, COMPRESSION,

ENCRYPTION (ALGORITHM = AES\_256, SERVER CERTIFICATE = AdventureWorks2019Cert)

GO

-- Restore database

RESTORE DATABASE AdventureWorks2019

FROM URL = 's3://<endpoint>:<port>/<bucket>/AdventureWorks2019\_Encrypt.bak'

WITH REPLACE

##### Use region for backup and restore.

The following example shows how to back up and restore the AdventureWorks2019 database using REGION\_OPTIONS:

-- Backup Database

BACKUP DATABASE AdventureWorks2019

TO URL = 's3://<endpoint>:<port>/<bucket>/AdventureWorks2019.bak'

WITH BACKUP\_OPTIONS = '{"s3": {"region":"us-east-1"}}'

-- Restore Database

RESTORE DATABASE AdventureWorks2019

FROM URL = 's3://<endpoint>:<port>/<bucket>/AdventureWorks2019.bak'

WITH MOVE 'AdventureWorks2019' TO 'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\AdventureWorks2019.mdf'

, MOVE 'AdventureWorks2019\_log' TO 'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\AdventureWorks2019.ldf'

, RESTORE\_OPTIONS = '{"s3": {"region":"us-east-1"}}'

### Additional reading

* [SQL Server backup to URL for S3-compatible object storage - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/relational-databases/backup-restore/sql-server-backup-to-url-s3-compatible-object-storage?view=sql-server-ver16)

## Accelerated Database Recovery (ADR)

Accelerated database recovery (ADR) improves database availability, especially in the presence of long running transactions, by redesigning the SQL database engine recovery process. The new recovery mechanism combines traditional ARIES recovery with multi-version concurrency control to achieve database recovery in constant time, regardless of the size of user transactions. Additionally, our algorithm enables continuous transaction log truncation, even in the presence of long running transactions, thereby allowing large data modifications using only a small, constant amount of log space.

### What is Accelerated Database Recovery (ADR)?

Accelerated Database Recovery (ADR) is a new SQL database engine feature that greatly improves database availability, especially in the presence of long running transactions, by redesigning the SQL database engine recovery process. ADR is currently available for single databases and pooled databases in Azure SQL Database, and databases in Azure Synapse (previously SQL Data Warehouse), and SQL Server 2019 and SQL server 2022.

There are several improvements to address persistent version store (PVS) storage and improve overall scalability. SQL Server 2022 (16.x) implements a persistent version store cleaner thread per database instead of per instance and the memory footprint for PVS page tracker has been improved. There are also several ADR efficiency improvements, such as concurrency improvements that help the cleanup process to work more efficiently. ADR cleans pages that couldn't previously be cleaned due to locking.

### ADR improvements in SQL Server 2022 (16.x)

There are several improvements to address persistent version store (PVS) storage and improve overall scalability. For more information about new features of SQL Server 2022 (16.x),

* User transaction cleanup

Clear pages that could not be cleaned by the regular process due to lock failure. This feature allows user transactions to run cleanup on pages that could not be addressed by the regular cleanup process due to table level lock conflicts. This improvement will ensure that the ADR cleanup process does not fail indefinitely because user workloads cannot acquire table level locks.

* Reducing memory footprint for PVS page tracker

This improvement tracks persisted version store (PVS) pages at an extent level, to reduce the memory footprint needed to maintain versioned pages.

* Accelerated Data Recovery (ADR) Cleaner improvements.

Accelerated Data Recovery (ADR) cleaner has improved version cleanup efficiencies to improve how SQL Server tracks and records aborted versions of a page leading to improvements in memory and capacity.

* Transaction-level persisted version store (PVS)

This improvement allows ADR to clean up versions belonging to committed transactions independent of whether there are aborted transactions in the system. With this improvement persisted version store (PVS) pages can be deallocated, even if the cleanup cannot complete a successful sweep to trim the aborted transaction map.

The result of this improvement is reduced persisted version store (PVS) growth even if ADR cleanup is slow or fails.

* Multi-threaded version cleanup

In SQL Server 2019 (15.x), the cleanup process is single threaded within a SQL Server instance.

Beginning with SQL Server 2022 (16.x), this process uses multi-threaded version cleanup. This allows multiple databases in the same SQL Server instance to be cleaned in parallel. This improvement is valuable when you have multiple large databases.

To adjust the number of threads for version cleanup scalability, set ADR Cleaner Thread Count with sp\_configure. The thread count is capped at the number of cores for instance.

* **New extended event**

A new extended event, tx\_mtvc2\_sweep\_stats, has been added for telemetry on the ADR PVS multi-threaded version cleaner.

The primary benefits of ADR are:

##### Fast and consistent database recovery

With ADR, long running transactions do not impact the overall recovery time, enabling fast and consistent database recovery irrespective of the number of active transactions in the system or their sizes.

##### Instantaneous transaction rollback

With ADR, transaction rollback is instantaneous, irrespective of the time that the transaction has been active or the number of updates that has performed.

##### Aggressive log truncation

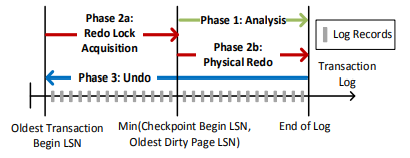
With ADR, the transaction log is aggressively truncated, even in the presence of active long running transactions, which prevents it from growing out of control.

### The Current Recovery Process (without ADR)

The recovery process of a database can be a challenge. Not just with large databases, but smaller databases with heavy transactions as well. The main challenge is the time it takes to bring a database online. The recovery requires a rollback of all incomplete transactions. The length of time required is proportional to the work that the transaction has performed and the time it has been active. Therefore, the SQL Server recovery process can take a long time in the presence of long running transactions, such as large bulk insert operations or index build operations against a large table.

Cancelling/rolling back a large transaction based on this design can also take a long time as it is using the same Undo recovery phase as described above.

In addition, the SQL database engine cannot truncate the transaction log when there are long running transactions because their corresponding log records are needed for the recovery and rollback processes. As a result of this design of the SQL database engine, some customers face the problem that the size of the transaction log grows very large and consumes huge amounts of drive space.



This illustration shows how the three phases of the recovery process work.

1. **Analysis phase**  
   SQL Server conducts a forward scan of the transaction log from the beginning of the last successful checkpoint (or the oldest dirty page LSN) until the end, to determine the state of each transaction at the time SQL Server stopped.
2. **Redo phase**  
   SQL Server performs forward scan of the transaction log from the oldest uncommitted transaction until the end, to bring the database to the state it was at the time of the crash by redoing all committed operations.
3. **Undo phase**  
   For each transaction that was active as of the time of the crash, SQL Server traverses the log backwards, undoing the operations that this transaction performed.

Based on this design, the time it takes the database engine to recover from an unexpected restart is roughly proportional to the size of the longest active transaction.

### Changes with ADR

ADR addresses the issues of the normal recovery process by completely redesigning the SQL database engine recovery process to:

* Make it constant time/instant by avoiding having to scan the log from/to the beginning of the oldest active transaction.
* With ADR, the transaction log is only processed from the last successful checkpoint (or oldest, dirty page Log Sequence Number (LSN)). As a result, recovery time is not impacted by long running transactions.
* Minimize the required transaction log space since there is no longer a need to process the log for the whole transaction. As a result, the transaction log can be truncated aggressively as checkpoints and backups occur.
* At a High Level, ADR achieves fast database recovery by versioning all physical database modifications and only undoing logical operations, which are limited and can be undone almost instantly.
* Any transaction that was active as of the time of a crash are marked as aborted and, therefore, any versions generated by these transactions can be ignored by concurrent user queries.

##### Persistent Version Store

At the heart of these changes is the Persistent Version Store (PVS). PVS allows row versions to be recoverable by storing them in the user database and logging them in the transaction log as regular user data. Hence, at the end of Redo all versions are fully recovered and can be accessed by user transactions. These versions are needed for recovery purposes and must be preserved until the committed version of each row has been brought back to the data page, via a background cleanup process.

##### Logical Revert

Logical revert is the asynchronous process (by a background cleanup process) of bringing the committed version of a row back to the main row in the data page, so that all queries can access it directly and versions in the version store are no longer required. This process compares the state of the aborted and committed versions and performs the required compensating operation (insert, update, or delete) to get the row to the committed state.

With this mechanism, both reads and writes can access or update any row immediately after a transaction that updated it rolls back. The same process applies during recovery, eliminating the costly Undo process that undoes each operation performed by uncommitted transactions.

##### Slog

SQL Server has a wide variety of operations that cannot be versioned because they are:

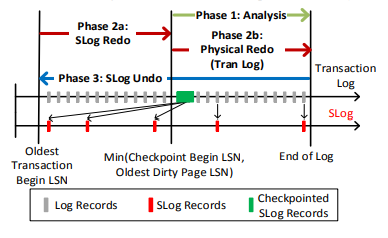
* Logical, such as acquiring coarse-grained locks at the table or index level, invalidating various caches when a transaction rolls back or accumulating row and page statistics for Bulk operations.
* Updating system metadata in data structures that are highly compacted, such as information about which pages are allocated.
* Updating critical system metadata required for starting up the database, before recovery can reconstruct versioning information, such as updates to the “boot page”, a special page that contains the core information required for initialization.

Slog is a secondary log stream designed to only track non-versioned operations that must be redone or undone using information from the corresponding log records. This allows us to efficiently process relevant log records without having to scan the full transaction log. Given that such operations are generally associated with schema changes and other rare database operations, such as changing various database options, the volume of log records written to the Slog is several orders of magnitude lower than the total volume of the transaction log, allowing us to process these operations in minimal time.

Slog in-memory log stream that is persisted to disk by being serialized into the traditional transaction log. In memory, Slog is stored as a linked list of log records, ordered based on their LSNs to allow efficient traversal from the oldest to the newest LSN during Redo and Undo.

### New Recovery Process with ADR

While the above descriptions of the Persistent Version Store (PVS) and the Slog are simplified, we now have a basic understanding of the changes and can now see how the recovery process improves.



1. **Analysis phase**  
   The process remains the same as today with the addition of reconstructing Slog and copying log records for non-versioned operations.
2. **Redo phase**  
   Broken into two sub-phases:
   1. **Sub-phase 1**  
      Redo from Slog (oldest uncommitted transaction up to last checkpoint). Redo is a fast operation as it only needs to process a few records from the Slog.
   2. **Sub phase 2**  
      Redo from transaction log starts from last checkpoint (instead of oldest uncommitted transaction).
3. **Undo phase**  
   The undo phase with ADR completes almost instantaneously by using Slog to undo non-versioned operations and persisted version store (PVS) with logical revert to perform row level version-based undo.

### New Overhead to Consider

##### Larger Data File

The data file is larger, but on the other hand ADR enabled databases will have aggressive log truncation so log files will always be small. Additionally, logging the versions introduces a performance overhead for the user transactions that generate them.

The background cleanup process helps address the additional space usage. This process is responsible for deleting earlier versions and freeing up space in the database.

##### Performance Impact

Heavy OLTP workloads are more susceptible to performance impacts. ADR clearly introduces some overhead by logging the generated versions, checking for current versions, and the cleanup processes itself. Once the cleanup process is completed, however, the performance overhead for queries that traverse multiple versions to access the committed version of each row is now eliminated.

### Using ADR

### What Workloads can Benefit from ADR?

Not all workloads are well suited for ADR. ADR improves database availability, especially in the presence of long running transactions. This is the ideal workload. You may also benefit from ADR if your recovery times are longer than your SLA.

Very heavy OLTP workloads, particularly workloads with heavy UPDATE operations, are more susceptible to degraded performance. This is because many updates cause multiple versions in the PVS, creating performance overhead for user queries to traverse multiple versions to get to the committed version.

As with many features, we recommend testing the feature with a workload before turning it on.

### Turning ADR On

The syntax for turning on ADR is just a simple ALTER DATABASE command.

ALTER DATABASE [MyDatabase]   
SET ACCELERATED\_DATABASE\_RECOVERY = ON;

CAUTION: It is not recommended to set ACCELERATED\_DATABASE\_RECOVERY ON and OFF repeatedly over a short period of time.

##### Force Manual Cleanup

ADR automatically cleans up the versions in the data file, but you can force it to cleanup manually, with persistent\_version\_cleanup stored procedure. This procedure is primarily used for moving the PVS to a different filegroup, as discussed below.

EXEC sys.sp\_persistent\_version\_cleanup [MyDatabase]

### Persistent Version Store Size

You can view the size of PVS by querying the sys.dm\_persistent\_version\_store\_stats DMV. This may be a good measure to capture, so you can monitor overtime, along with overall database size because these numbers will change.

SELECT DB\_Name(database\_id),   
 persistent\_version\_store\_size\_kb   
FROM sys.dm\_tran\_persistent\_version\_store\_stats  
WHERE database\_id = [MyDatabaseID]

PVS is considered large if it's significantly larger than baseline or if it is close to 50% of the size of the database.

### Change Location of the PVS to a Different Filegroup

Please refer to the published instructions at the link below.

<https://docs.microsoft.com/en-us/sql/relational-databases/accelerated-database-recovery-management#change-the-location-of-the-pvs-to-a-different-filegroup>

# Database Scoped Configurations

Starting from SQL Server 2016 several SQL server level configurations were made available at database level. When you write click the database and go to the properties you will see them under Options tab. In 2022 the list now contains 35 database scope configurations and is no longer made available in SSMS GUI. But you can user alter database command to turn on and off these configurations.

## Database Scoped Configs by Version

**SQL Server 2016 (4)**

1. MAXDOP
2. LEGACY\_CARDINALITY\_ESTIMATION
3. PARAMETER\_SNIFFING
4. QUERY\_OPTIMIZER\_HOTFIXES

**SQL Server 2017 (5)**

1. MAXDOP
2. LEGACY\_CARDINALITY\_ESTIMATION
3. PARAMETER\_SNIFFING
4. QUERY\_OPTIMIZER\_HOTFIXES
5. IDENTITY\_CACHE

**SQL Server 2019 (23)**

1. MAXDOP
2. LEGACY\_CARDINALITY\_ESTIMATION
3. PARAMETER\_SNIFFING
4. QUERY\_OPTIMIZER\_HOTFIXES
5. IDENTITY\_CACHE
6. INTERLEAVED\_EXECUTION\_TVF
7. BATCH\_MODE\_MEMORY\_GRANT\_FEEDBACK
8. BATCH\_MODE\_ADAPTIVE\_JOINS
9. TSQL\_SCALAR\_UDF\_INLINING
10. ELEVATE\_ONLINE
11. ELEVATE\_RESUMABLE
12. OPTIMIZE\_FOR\_AD\_HOC\_WORKLOADS
13. XTP\_PROCEDURE\_EXECUTION\_STATISTICS
14. XTP\_QUERY\_EXECUTION\_STATISTICS
15. ROW\_MODE\_MEMORY\_GRANT\_FEEDBACK
16. ISOLATE\_SECURITY\_POLICY\_CARDINALITY
17. BATCH\_MODE\_ON\_ROWSTORE
18. DEFERRED\_COMPILATION\_TV
19. ACCELERATED\_PLAN\_FORCING
20. GLOBAL\_TEMPORARY\_TABLE\_AUTO\_DROP
21. LIGHTWEIGHT\_QUERY\_PROFILING
22. VERBOSE\_TRUNCATION\_WARNINGS
23. LAST\_QUERY\_PLAN\_STATS

**SQL SERVER 2022 (35)**

1. MAXDOP
2. LEGACY\_CARDINALITY\_ESTIMATION
3. PARAMETER\_SNIFFING
4. QUERY\_OPTIMIZER\_HOTFIXES
5. IDENTITY\_CACHE
6. INTERLEAVED\_EXECUTION\_TVF
7. BATCH\_MODE\_MEMORY\_GRANT\_FEEDBACK
8. BATCH\_MODE\_ADAPTIVE\_JOINS
9. TSQL\_SCALAR\_UDF\_INLINING
10. ELEVATE\_ONLINE
11. ELEVATE\_RESUMABLE
12. OPTIMIZE\_FOR\_AD\_HOC\_WORKLOADS
13. XTP\_PROCEDURE\_EXECUTION\_STATISTICS
14. XTP\_QUERY\_EXECUTION\_STATISTICS
15. ROW\_MODE\_MEMORY\_GRANT\_FEEDBACK
16. ISOLATE\_SECURITY\_POLICY\_CARDINALITY
17. BATCH\_MODE\_ON\_ROWSTORE
18. DEFERRED\_COMPILATION\_TV
19. ACCELERATED\_PLAN\_FORCING
20. GLOBAL\_TEMPORARY\_TABLE\_AUTO\_DROP
21. LIGHTWEIGHT\_QUERY\_PROFILING
22. VERBOSE\_TRUNCATION\_WARNINGS
23. LAST\_QUERY\_PLAN\_STATS
24. PAUSED\_RESUMABLE\_INDEX\_ABORT\_DURATION\_MINUTES
25. DW\_COMPATIBILITY\_LEVEL
26. EXEC\_QUERY\_STATS\_FOR\_SCALAR\_FUNCTIONS
27. PARAMETER\_SENSITIVE\_PLAN\_OPTIMIZATION
28. ASYNC\_STATS\_UPDATE\_WAIT\_AT\_LOW\_PRIORITY
29. CE\_FEEDBACK
30. MEMORY\_GRANT\_FEEDBACK\_PERSISTENCE
31. MEMORY\_GRANT\_FEEDBACK\_PERCENTILE\_GRANT
32. OPTIMIZED\_PLAN\_FORCING
33. DOP\_FEEDBACK
34. LEDGER\_DIGEST\_STORAGE\_ENDPOINT
35. FORCE\_SHOWPLAN\_RUNTIME\_PARAMETER\_COLLECTION

Use the sys.database\_scoped\_configurations system view to see the full list.

SELECT \* FROM sys.database\_scoped\_configurations;

For more information see, [sys-databases-transact-sql | SQL Docs](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-databases-transact-sql).

# Intelligent Query Processing (IQP)

The intelligent query processing (IQP) feature family includes features with broad impact that improve the performance of existing workloads with minimal implementation effort to adopt.

You can make workloads automatically eligible for intelligent query processing by enabling the applicable database compatibility level for the database.

ALTER DATABASE [WideWorldImportersDW] SET COMPATIBILITY\_LEVEL = 160;

The blue squares in the image below illustrate IQP features added in SQL Server 2022.

A picture containing text, screenshot, font, diagram

Description automatically generated

### Migration Guidance

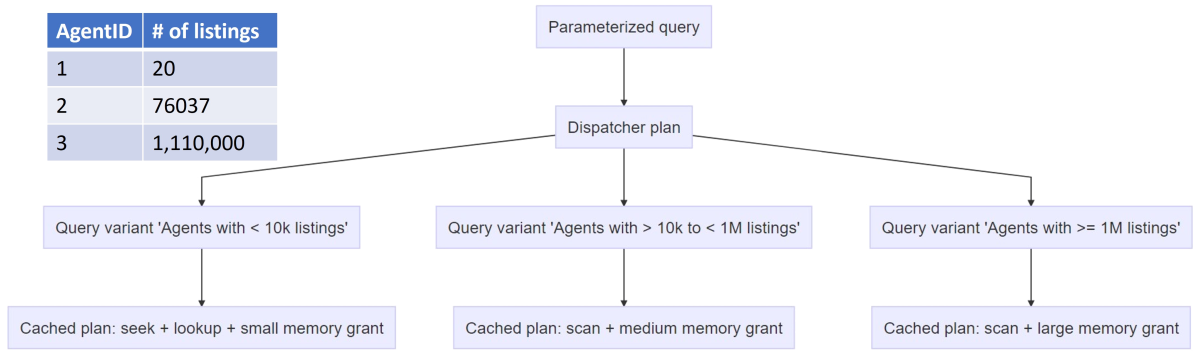
When migrating to SQL Server 2022, we recommend disabling IQP features initially and testing each feature in a non-production environment first.

## Parameter sensitive plan optimization

Parameter Sensitive Plan (PSP) optimization is part of the [Intelligent query processing](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?view=sql-server-ver16) family of features. It addresses the scenario where a single cached plan for a parameterized query isn't optimal for all possible incoming parameter values. This is the case with non-uniform data distributions. Automatically enables multiple, active cached plans for a single parameterized statement. Cached execution plans accommodate largely different data sizes based on the customer-provided runtime parameter value(s).

### Predicate cardinality range

At runtime, the cardinality of each predicate is evaluated based on runtime parameter values. The dispatcher bucketizes the cardinality values into three predicate cardinality ranges at compile time. For example, the PSP optimization feature can create three ranges that would represent low, medium, and high cardinality ranges, as shown in the following diagram.



In other words, when a parameterized query is initially compiled, the PSP optimization feature generates a shell plan known as a dispatcher plan. The dispatcher expression has the logic that bucketizes queries into query variants based on the runtime values of parameters. When actual execution begins, the dispatcher performs two steps:

* The dispatcher evaluates its dispatcher expression for the given set of parameters to compute the cardinality range.
* The dispatcher maps these ranges to specific query variants and compiles and executes the variants. By virtue of having multiple query variants, the PSP optimization feature achieves having multiple plans for a single query.

See more information [Parameter Sensitive Plan optimization - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/relational-databases/performance/parameter-sensitive-plan-optimization?view=sql-server-ver16)

## Degree of Parallelism (DOP) feedback

SQL Server 2022 (16.x) introduced a new feature called degree of parallelism (DOP) feedback to improve query performance by identifying parallelism inefficiencies for repeating queries, based on elapsed time and waits. DOP feedback is part of the [intelligent query processing](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?view=sql-server-ver16) family of features, and addresses suboptimal usage of parallelism for repeating queries. This scenario helps with optimizing resource usage and improving the scalability of workloads when excessive parallelism can cause performance issues. Instead of incurring in the pains of an all-encompassing default or manual adjustments to each query, DOP feedback self-adjusts DOP to avoid the issues described above.

Starting with SQL 2022, A new database scoped configuration option DOP\_FEEDBACK automatically adjusts degree of parallelism for repeating queries to optimize for workloads where inefficient parallelism can cause performance issues. Like optimizations in Azure SQL Database. Requires the Query Store to be enabled and in "Read write" mode.

Beginning with RC 0, every query recompilation SQL Server compares the runtime stats of the query using existing feedback to the runtime stats of the previous compilation with the existing feedback. If the performance isn't the same or better, we clear all DOP feedback and trigger a reanalysis of the query starting from the compiled DOP.

See [Query processing feedback features - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing-feedback?view=sql-server-ver16#degree-of-parallelism-dop-feedback)

### Migration Guidance

When migrating to SQL Server 2022, we recommend disabling this IQP feature initially and testing each feature in a non-production environment first.

Test your database in database compatibility level 160, with IQP features disabled at the database level. Once stabilized, turn on each IQP feature, one at a time, testing and stabilizing each.

## Cardinality estimation feedback

Starting with SQL Server 2022 (16.x)), the Cardinality Estimation (CE) feedback is part of the [intelligent query processing family of features](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?view=sql-server-ver16) and addresses suboptimal query execution plans for repeating queries when these issues result from incorrect CE model assumptions. This scenario helps with reducing regression risks related to the default CE when upgrading from older versions of the Database Engine.

Because no single set of CE models and assumptions can accommodate the vast array of customer workloads and data distributions, CE feedback provides an adaptable solution based on query runtime characteristics. CE feedback will identify and use a model assumption that better fits a given query and data distribution to improve query execution plan quality. Currently, CE Feedback can identify plan operators where the estimated number of rows and the actual number of rows are very different. Feedback is applied when significant model estimation errors occur, and there is a viable alternate model to try.

* Starting with SQL Server 2022 (16.x), when the Query Store for secondary replicas is enabled, CE feedback is not replica-aware for secondary replicas in availability groups. CE feedback currently only benefits primary replicas.

Identifies and corrects suboptimal query execution plans for repeating queries when these issues are caused by incorrect estimation model assumptions. Requires the Query Store to be enabled and in "Read write" mode.

### Understand Cardinality Estimation

Cardinality Estimation (CE) is how the Query Optimizer can estimate the total number of rows processed at each level of a query plan. Cardinality estimation in SQL Server is derived primarily from histograms created when indexes or statistics are created, either manually or automatically. Sometimes, SQL Server also uses constraint information and logical rewrites of queries to determine cardinality.

Different versions of the Database Engine use different CE model assumptions based on how data is distributed and queried. For more information, see [versions of the CE](https://learn.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server?view=sql-server-ver16#versions-of-the-ce).

### CE feedback implementation

CE feedback learns which CE model assumptions are optimal over time and then apply the historically most correct assumption:

1. CE feedback **identifies** model-related assumptions and evaluates whether they're accurate for repeating queries.
2. If an assumption looks incorrect, a subsequent execution of the same query is tested with a query plan that adjusts the impactful CE model assumption and **verifies** if it helps. We identify incorrectness by looking at actual vs. estimated rows from plan operators. Not all errors can be corrected by model variants available in CE feedback.
3. If it improves plan quality, the old query plan is **replaced** with a query plan that uses the appropriate [USE HINT query hint](https://learn.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query?view=sql-server-ver16#l-using-use-hint) that adjusts the estimation model, implemented through the [Query Store hint](https://learn.microsoft.com/en-us/sql/relational-databases/performance/query-store-hints?view=sql-server-ver16) mechanism.

Only verified feedback persists. CE feedback isn't used for that query if the adjusted model assumption results in a performance regression. In this context, a user canceled query is also perceived as a regression.

### CE feedback scenarios

CE feedback addresses perceived regression issues resulting from incorrect CE model assumptions when using the default CE (CE120 or higher) and can selectively use different model assumptions. The scenarios include Correlation, Join Containment, and Optimizer row goal.

**Correlation**

When the Query Optimizer estimates the selectivity of predicates on a given table or view, or the number of rows satisfying the said predicate, it uses **correlation model assumptions**. These assumptions can be that predicates are:

* **Fully independent** (default for CE70), where cardinality is calculated by multiplying the selectivity of all predicates.
* **Partially correlated** (default for CE120 and higher), where cardinality is calculated using a variation on exponential backoff, ordering the selectivity from most to the least selective predicate.
* **Fully correlated**, where cardinality is calculated by using the minimum selectivity for all predicates.

The following example uses partial correlation when the database compatibility is set to 120 or higher:

USE AdventureWorks2016\_EXT;

GO

SELECT AddressID, AddressLine1, AddressLine2

FROM Person. Address

WHERE StateProvinceID = 79 AND City = N'Redmond';

GO

When the database compatibility is set to 160, and default correlation is used, CE feedback will attempt to move the correlation to the correct direction one step at a time based on whether the estimated cardinality was underestimated or overestimated compared to the actual number of rows. Use full correlation if an actual number of rows is greater than the estimated cardinality. Use full independence if an actual number of rows is smaller than the estimated cardinality.

### Considerations for CE feedback

To enable CE feedback, enable database compatibility level 160 for the database you're connected to when executing the query. The Query Store must be enabled and in READ\_WRITE mode for every database where CE feedback is used.

CE feedback activity is visible via the query\_feedback\_analysis and query\_feedback\_validation XEvents.

Hints set by CE feedback can be tracked using the [sys.query\_store\_query\_hints](https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-query-store-query-hints-transact-sql?view=sql-server-ver16) catalog view.

Feedback information can be tracked using the [sys.query\_store\_plan\_feedback](https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-query-store-plan-feedback?view=sql-server-ver16) catalog view.

To disable CE feedback at the database level, use the ALTER DATABASE SCOPED CONFIGURATION SET CE\_FEEDBACK = OFF [database scoped configuration](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-database-scoped-configuration-transact-sql?view=sql-server-ver16#ce_feedback---on--off-).

To disable CE feedback at the query level, use the DISABLE\_CE\_FEEDBACK query hint.

If a query has a query plan forced through the Query Store, CE feedback won't be used for that query.

If a query uses hard-coded query hints or is using Query Store hints set by the user, CE feedback won't be used for that query.

For more information see [Query processing feedback features - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing-feedback?view=sql-server-ver16#cardinality-estimation-ce-feedback)

## Memory Grant Feedback Highlights

Memory grant feedback adjusts the size of the memory allocated for a query based on past performance. SQL Server 2022 (16.x) introduces [Percentile and Persistence mode memory grant feedback](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing-feedback?view=sql-server-ver16#percentile-and-persistence-mode-memory-grant-feedback). Requires enabling Query Store.  
  
- Persistence: A capability that allows the memory grant feedback for a given cached plan to be persisted in the Query Store so that feedback can be reused after cache evictions. Persistence benefits memory grant feedback as well as the new DOP and CE feedback features.  
- Percentile: A new algorithm improves performance of queries with widely oscillating memory requirements, using memory grant information from several previous query executions over, instead of just the memory grant from the immediately preceding query execution. Requires enabling Query Store. Query Store is enabled by default for newly created databases as of SQL Server 2022 CTP 2.1

### Percentile and persistence mode memory grant feedback

This feature was introduced in SQL Server 2022 (16.x), however this performance enhancement is available for queries that operate in the database compatibility level 140 (introduced in SQL Server 2017) or higher, or the QUERY\_OPTIMIZER\_COMPATIBILITY\_LEVEL\_n hint of 140 and higher, and when the Query Store is enabled for the database and is in a "read write" state.

Percentile memory grant feedback is enabled by default in SQL Server 2022 (16.x) but has no effect if Query Store is not enabled or when Query Store is not in a "read write" state.

Persistence for memory grant, CE, and DOP feedback is on by default in SQL Server 2022 (16.x) but has no effect when Query Store is not enabled or when Query Store is not in a "read write" state.

Percentile and persistence for memory grant feedback is available in Azure SQL Database, and enabled by default on all databases, both existing and new.

Percentile and persistence for memory grant feedback is not currently available in Azure SQL Managed Instance.

It's recommended that you have a performance baseline for your workload before the feature is enabled for your database. The baseline numbers will help you determine if you're getting the intended benefit from the feature.

Memory grant feedback (MGF) is an existing feature that adjusts the size of the memory allocated for a query based on past performance. However, the initial phases of this project only stored the memory grant adjustment with the plan in the cache – if a plan is evicted from the cache, the feedback process must start again, resulting in poor performance the first few times a query is executed after eviction. The new solution is to persist the grant information with the other query information in the Query Store so that the benefits last across cache evictions. Memory grant feedback persistence and percentile address existing limitations of memory grant feedback in a non-intrusive way.

Additionally, the grant size adjustments only accounted for the most recently used grant. So, if a parameterized query or workload requires significantly varying memory grant sizes with each execution, the most recent grant information could be inaccurate. It could be out of step with the actual needs of the query being executed. Memory grant feedback in this scenario is unhelpful to performance because we're always adjusting memory based on the last used grant value. The next image shows the behavior possible with memory grant feedback without percentile and persistence mode.

A picture containing line, diagram, plot

Description automatically generated

As you can see, in this unusual but possible query behavior, the oscillation between the actual needed and granted memory amounts results in wasted and insufficient memory if the query execution itself alternates in terms of the amount of memory. In this scenario, memory grant feedback disables itself, recognizing it's doing more harm than good.

Using a percentile-based calculation over recent history of the query, instead of simply the last execution, we can smooth the grant size values based on past execution usage history and try to optimize for minimizing spills. For example, the same alternating workload would see the following memory grant behavior:

A picture containing line, diagram

Description automatically generated

The query optimizer uses a high percentile of past memory grant sizing requirements for executions of the cached plan to calculate memory grant sizes, using data persisted in the Query Store. The percentile adjustment, which will perform the memory grant adjustments is based on the recent history of executions. Over time, the memory grant given reduces spills and wasted memory.

### Enable memory grant feedback: persistence and percentile.

Persistence and percentile feedback are [**enabled by default**](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing-feedback?view=sql-server-ver16#percentile-and-persistence-mode-memory-grant-feedback) in Azure SQL Database and SQL Server 2022 (16.x).

Use database compatibility level 140 or higher for the database you're connected to when executing the query. You can change this via [**ALTER DATABASE**](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql-compatibility-level?view=sql-server-ver16):

ALTER DATABASE <DATABASE NAME> SET COMPATIBILITY LEVEL = 140; -- OR HIGHER

The Query Store must be enabled for every database where the persistence portion of this feature is used.

### Disable percentile.

To disable memory grant feedback percentile for all query executions originating from the database, execute the following within the context of the applicable database:

ALTER DATABASE SCOPED CONFIGURATION SET MEMORY\_GRANT\_FEEDBACK\_PERCENTILE = OFF;

The default setting for MEMORY\_GRANT\_FEEDBACK\_PERCENTILE is ON.

### Disable persistence.

To disable memory grant feedback persistence for all query executions originating from the database.

Execute the following within the context of the applicable database:

ALTER DATABASE SCOPED CONFIGURATION SET MEMORY\_GRANT\_FEEDBACK\_PERSISTENCE = OFF;

### Disabling memory grant feedback persistence will also remove existing collected feedback.

### The default setting for MEMORY\_GRANT\_FEEDBACK\_PERSISTENCE is ON.

### Considerations for memory grant feedback

Given feedback data is now persisted in the Query Store, there's some increase in the Query Store usage requirements.

Percentile-based memory grant errs on the side of reducing spills. Because it's no longer based on the last execution-only but on an observation of the several past executions, this could increase memory usage for oscillating workloads with wide variance in memory grant requirements between executions.

Starting with SQL Server 2022 (16.x), when Query Store for secondary replicas is enabled, memory grant feedback is replica-aware for secondary replicas in availability groups. Memory grant feedback can apply feedback differently on a primary replica and on a secondary replica. However, memory grant feedback is not persisted on secondary replicas, and on failover, the memory grant feedback from the old primary replica is applied to the new primary replica. Any feedback applied to the secondary replica when it becomes the primary replica is lost.

### Migration Guidance

When migrating to SQL Server 2022, we recommend disabling this IQP feature initially and testing each feature in a non-production environment first.

Test your database in database compatibility level 160, with IQP features disabled at the database level. Once stabilized, turn on each IQP feature, one at a time, testing and stabilizing each.

## Optimized plan forcing

Query optimization is a multi-phased process of generating a "good-enough" query execution plan. In some cases, query compilation, a part of query optimization, can represent a large percentage of overall query execution time and consume significant system resources. Optimized plan forcing is part of the [intelligent query processing family of features](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?view=sql-server-ver16). Optimized plan forcing reduces compilation overhead for repeating forced queries and requires the Query Store to be enabled and in "read write" mode. Once the query execution plan is generated, specific compilation steps are stored for reuse as an optimization replay script. An optimization replay script is stored as part of the compressed showplan XML in [Query Store](https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver16), in a hidden OptimizationReplay attribute.

### Optimized plan forcing implementation

When a query first goes through the compilation process, a threshold based on estimation of the time spent in optimization (based on the query optimizer input tree) will determine whether an optimization replay script is created.

After compilation is completed, several runtime metrics become available to assess whether the previous estimation was correct. If it's confirmed the threshold was crossed, the optimization replay script is eligible for persistence. These runtime metrics include the number of objects accessed, the number of joins, the number of optimization tasks executed during optimization, and the actual optimization time.

The potential benefit of using an optimization replay script is also compared to the overhead of storing the optimization replay script. An estimation of the relative time to replay the optimization replay script is compared with the time that was spent executing the normal optimization process, based on the number of optimization tasks stored in optimization replay script and the number of optimization tasks executed during normal compilation. If replaying the optimization replay script shows substantial benefit in reducing compilation time, the optimization replay script is persisted.

### Considerations

When the optimized plan forcing feature is enabled, the eligibility criteria for optimized plan forcing is:

1. Only query plans that go through full optimization are eligible, which can be verified by the presence of the StatementOptmLevel="FULL" property.
2. Statements with RECOMPILE hint and distributed queries are not eligible.

However, if the Query Store independently captures a query plan that was scoped out by optimized plan forcing, the optimization replay script will be created for a second recompilation of that same query, subject to default recompilation events. Learn more about recompilation in [Recompiling Execution Plans](https://learn.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver16#recompiling-execution-plans).

Even if an optimization replay script was generated, it might not be persisted in the Query Store if the Query Store configured capture policies criteria isn't met, notably the number of executions of that statement and its cumulated compile and execution times. In this case, the invalid optimization replay script will be removed from memory asynchronously.

##### Enable and disable optimized plan forcing.

You can enable or disable optimized plan forcing for a database. When optimized plan forcing is enabled for a database, you may disable it for individual queries using the DISABLE\_OPTIMIZED\_PLAN\_FORCING query hint. You may also disable optimized plan forcing for a query plan which is forced in Query Store.

##### Enable or disable optimized plan forcing a database.

Optimized plan forcing is enabled by default for new databases created in SQL Server 2022 (16.x) and higher. The Query Store must be enabled for every database where optimized plan forcing is used. Upgraded instances with existing databases or databases restored from a lower version of SQL Server will have optimized plan forcing enabled by default.

To enable optimized plan forcing at the database level, use the ALTER DATABASE SCOPED CONFIGURATION SET OPTIMIZED\_PLAN\_FORCING = ON database scoped configuration. You must enable the Query Store if it isn't already enabled.

To disable optimized plan forcing at the database level, use the ALTER DATABASE SCOPED CONFIGURATION SET OPTIMIZED\_PLAN\_FORCING = OFF database scoped configuration.

##### Disable optimized plan forcing with a query hint.

When the optimized plan forcing feature is enabled in a database, you can disable optimized plan forcing for an individual query by using the DISABLE\_OPTIMIZED\_PLAN\_FORCING [query hint](https://learn.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query?view=sql-server-ver16).

##### Force a plan with the Query Store but disable optimized plan forcing.

The [sp\_query\_store\_force\_plan](https://learn.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-query-store-force-plan-transact-sql?view=sql-server-ver16) procedure includes a disable\_optimized\_plan\_forcing parameter. To use this parameter, an additional parameter is required by the sp\_query\_store\_force\_plan stored procedure. The additional parameter is called replica\_group\_id. By default, the primary replica\_group\_id will have a value of one (*1*) even in case where there are no configured secondary replicas.

The sys. query\_store\_plan catalog view includes columns that indicate if the plan has an associated optimization replay script and adds a new state to existing failure reason column specific to associated optimization replay script.

##### Enable Query Store and optimized plan forcing on a database.

The following code enables Query Store on a database, then enables optimized plan forcing on the database. Learn more about options enabling Query Store in [ALTER DATABASE SET options (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql-set-options?view=sql-server-ver16).

Before running the code, connect to the appropriate user database.

A screenshot of a computer

Description automatically generated with medium confidence

##### Select all queries where optimized plan forcing is disabled by Query Store

The following example queries all plans that have been forced in Query Store where is\_optimized\_plan\_forcing\_disabled has been set to 1. Before running the code, connect to the appropriate user database.

A screenshot of a computer code

Description automatically generated with low confidence

See more information [Optimized plan forcing with Query Store - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/relational-databases/performance/optimized-plan-forcing-query-store?view=sql-server-ver16)

### Migration Guidance

When migrating to SQL Server 2022, we recommend disabling this IQP feature initially and testing each feature in a non-production environment first.

Test your database in database compatibility level 160, with IQP features disabled at the database level. Once stabilized, turn on each IQP feature, one at a time, testing and stabilizing each.

# Performance

## Index Updates

Some changes to index creation have been added to SQL Server 2022.

### WAIT\_AT\_LOW\_PRIORITY with online index operations.

**Staring SQL 2022,** [wait\_at\_low\_priority](https://learn.microsoft.com/en-us/sql/t-sql/statements/create-index-transact-sql?view=sql-server-ver16#wait-at-low-priority) with online index operations clause added.

This syntax for CREATE INDEX currently applies to SQL Server 2022 (16.x), Azure SQL Database, and Azure SQL Managed Instance only. For ALTER INDEX, this syntax applies to SQL Server (Starting with SQL Server 2014 (12.x)) and Azure SQL Database.

The low\_priority\_lock\_wait syntax allows for specifying WAIT\_AT\_LOW\_PRIORITY behavior. WAIT\_AT\_LOW\_PRIORITY can be used with ONLINE=ON only.

The WAIT\_AT\_LOW\_PRIORITY option allows DBAs to manage the Sch-S and Sch-M locks required for online index creation and allows them to select one of 3 options. In all 3 cases, if during the wait time MAX\_DURATION = n [minutes], there are no blocking activities, the online index rebuild is executed immediately without waiting and the DDL statement is completed.

WAIT\_AT\_LOW\_PRIORITY indicates that the online index create operation will wait for low priority locks, allowing other operations to proceed while the online index build operation is waiting. Omitting the WAIT AT LOW PRIORITY option is equivalent to WAIT\_AT\_LOW\_PRIORITY (MAX\_DURATION = 0 minutes, ABORT\_AFTER\_WAIT = NONE).

MAX\_DURATION = time [**MINUTES**]

The wait time (an integer value specified in minutes) that the online index creates locks will wait with low priority when executing the DDL command. If the operation is blocked for the MAX\_DURATION time, the specified ABORT\_AFTER\_WAIT action will be executed. MAX\_DURATION time is always in minutes, and the word **MINUTES** can be omitted.

ABORT\_AFTER\_WAIT = [**NONE** | **SELF** | **BLOCKERS**}]

NONE Continue waiting for the lock with normal (regular) priority.

SELF Exit the online index create DDL operation currently being executed, without taking any action. The option **SELF** can't be used with a MAX\_DURATION of 0.

BLOCKERS Kill all user transactions that block the online index rebuild DDL operation so that the operation can continue. The **BLOCKERS** option requires the login to have ALTER ANY

CONNECTION permission.

### Ordered clustered columnstore index.

Ordered clustered columnstore index (CCI) sorts the existing data in memory before the index builder compresses the data into index segments. This has the potential for more efficient segment elimination, resulting in better performance as the number of segments to read from disk is reduced. Ordered clustered columnstore indexes improve performance for queries based on ordered column predicates. Ordered columnstore indexes can improve performance by skipping segments of data altogether. This can drastically reduce IO needed to complete queries on columnstore data. Ordered cluster columnstore indexes are available in SQL Server 2022 (16.x).

Predicate pushdown with clustered columnstore row group elimination of strings uses boundary values to optimize string searches. All columnstore indexes benefit from enhanced segment elimination by data type. Starting with SQL Server 2022 (16.x), these segment elimination capabilities extend to string, binary, and guide data types, and the datetimeoffset data type for scale greater than two. Previously, columnstore segment elimination applied only to numeric, date, and time data types, and the datetimeoffset data type with scale less than or equal to two. After upgrading to a version of SQL Server that supports string min/max segment elimination (SQL Server 2022 (16.x) and later), the columnstore index will not benefit from this feature until it is rebuilt using a REBUILD or DROP/CREATE.

##### DROP\_EXISTING = [OFF] | ON

DROP\_EXISTING = ON specifies to drop the existing index and create a new columnstore index.

CREATE CLUSTERED COLUMNSTORE INDEX cci ON Sales.OrderLines

WITH (DROP\_EXISTING = ON);

The default, DROP\_EXISTING = OFF, expects the index name is the same as the existing name. An error occurs if the specified index name already exists.

### Improved columnstore segment elimination

All columnstore indexes benefit from enhanced segment elimination by data type. Data type choices may have a significant impact on query performance based on common filter predicates for queries on the columnstore index. This segment elimination applied to numeric, date, and time data types, and the datetimeoffset data type with scale less than or equal to two. Beginning in SQL Server 2022 (16.x), segment elimination capabilities extend to string, binary, guide data types, and the datetimeoffset data type for scale greater than two.

### Additional Reading

* [Guidance for Online Index Operations | SQL Docs](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/guidelines-for-online-index-operations)
* [CREATE INDEX (Transact-SQL) - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/t-sql/statements/create-index-transact-sql?view=sql-server-ver16#wait-at-low-priority)
* [What's new in columnstore indexes - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-what-s-new?view=sql-server-ver16)
* [Performance tuning with ordered clustered columnstore index - Azure Synapse Analytics | Microsoft Learn](https://learn.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/performance-tuning-ordered-cci)

## BufferPool Parallel Scan​

A buffer pool scan occurs when certain operations, such as database startup, shutdown, backup, restoration, or error checking, trigger the need to examine the contents of the buffer pool. During a scan, SQL Server searches through the buffer pool to locate specific pages or perform necessary operations.​

In SQL Server 2019 and previous releases, operations that require scanning the buffer pool can be slow, especially on large memory machines such as the M-series Azure SQL virtual machine and large on-premises SQL Server environments. Even log restore operations and availability group failover operations can be impacted. Currently, there’s no way to eliminate this issue prior to SQL Server 2022, and dropping buffers using DBCC DROPCLEANBUFFERS would likely result in some degree of performance degradation as any subsequent query executions will have to reread the data from the database files increasing I/O.​

As high-end machines with very large memory (i.e., tera-bytes) are more common for SQL Server and Azure SQL DB, customers have reported long delay in operations that require Buffer Pool scan [1]. Those operations include database creation/redo/restart, checkpoint, backup/restore, DBCC CheckDB, etc.

​

Some of the operations that flush dirty pages are done more efficiently by using Dirty Page Manager (which was introduced for indirect checkpoint IN SQL server 2016) that keeps track of dirty pages per database.​

However, there are still many operations and certain conditions that do entire buffer pool scan, which is fundamentally slow in a large buffer pool space. ​

Most of the cases fall into two underlying operations: flushing dirty buffers (FlushCache) and removing buffers from a database/file.​

For example, database restart (in case of CU update on AG replica) and dropping a data file require buffer pool scan to deallocate buffers. ​

Some backup and restore operations still rely on buffer pool scans to flush buffers.​

​

Here are some operations that may trigger a buffer pool scan to occur:​

* Database startup​
* Database shutdown or restart​
* AG failover​
* Database removal (drop)​
* File removal from a database​
* Full or differential database backup​
* Database restoration​
* Transaction log restoration​
* Online restoration​
* DBCC CHECKDB or DBCC CHECKTABLE operation​

**Workaround**​

Prior to SQL Server 2022, there was no way to eliminate this problem. It is not recommended to perform any action to clear the buffer pool as dropping clean buffers ([DBCC DROPCLEANBUFFERS](https://learn.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-dropcleanbuffers-transact-sql)) from the buffer pool may result in a significant performance degradation. Removing database pages from memory will cause subsequent query executions to re-read the data from the database files on disk. This process of accessing data via disk I/O causes queries to be slow.​

In SQL Server 2022, this problem is mitigated because buffer pool scans are parallelized by utilizing multiple cores. There will be one task per 8 million buffers (64 GB) where a serial scan will still be used if there are less than 8 million buffers.​

​

### Benefits of BufferPool​ Parallel Scan

* Buffer Pool parallel scan improves the performance of scan operations on large-memory machines by utilizing multiple CPU cores.​
* Customers running SQL Server on large-memory machines (data warehouse, OLTP) will witness faster executions scenarios which were historically slower due to the serialized buffer pool scan.
* The parallel scan feature also improves the buffer pool scan performance of small databases residing on large-memory machines. Additionally, this improvement adds buffer pool scan diagnostics and telemetry for supportability.​​
* Operations that scan the buffer pool, especially on servers with large amounts of memory, will benefit the most from the new SQL Server 2022 Buffer Pool Parallel Scan capability.​
* database startup/shutdown, creating a new database, file drop operations, backup/restore operations, Always On failover events, DBCC CHECKDB and DBCC Check Table, log restore operations, and other internal operations (e.g., checkpoint) will all benefit from Buffer Pool Parallel Scan.​
* Prior to 2022 scanning a buffer scan was always a serial operation​
* Improves adds buffer pool scan diagnostics and telemetry for supportability and insights​.
* Long visible scans will be visible by error logs if they take more than 10 seconds​.
* Extended events will capture scan start/complete, errors, flushcache.

### Example

A picture containing text, screenshot, font, number

Description automatically generated

### Additional Reading

* [Operations that trigger buffer pool scan may run slowly on large-memory computers - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-US/troubleshoot/sql/database-engine/performance/buffer-pool-scan-runs-slowly-large-memory-machines)​
* [Improve scalability with Buffer Pool Parallel Scan in SQL Server 2022 - Microsoft SQL Server Blog](https://cloudblogs.microsoft.com/sqlserver/2022/07/07/improve-scalability-with-buffer-pool-parallel-scan-in-sql-server-2022/#:~:text=In%20SQL%20Server%202022%2C%20the%20Buffer%20Pool%20Parallel,previously%20slower%20due%20to%20serialized%20buffer%20pool%20scans.)
* [SQL Server 2022: Introducing Buffer Pool Parallel Scan (Ep. 5) | Data Exposed - Microsoft Community Hub](https://techcommunity.microsoft.com/t5/video-hub/sql-server-2022-introducing-buffer-pool-parallel-scan-ep-5-data/ba-p/3200444)

## TEMPDB

Over the past several SQL Server releases, Microsoft has improved the concurrency and performance of the **tempdb**database. In [SQL Server 2022](https://info.microsoft.com/ww-landing-sql-server-2022.html) we are addressing one of the last areas of contention by introducing concurrent global allocation map (GAM) and shared global allocation map (SGAM) updates which will give SQL Server 2022 a big improvement for scalability as **tempdb**is arguably the most important database in your environment.

Tempdb performance challenges

Historically, tempdb has been one of those common pain points in SQL Server. Why was it a pain point? Well, usage is one of the key reasons. By usage, we are referring to creating temp tables and other user objects, but tempdb is also used internally to spill to disk when there isn’t enough memory available for a process or there is an inaccurate estimate that causes SQL Server to spill to tempdb.

##### What is the tempdb database?

Tempdb is a special purpose system database, but the structure is essentially just like any other user database. As the name suggests, the tempdb database was designed for temporary storage meaning that nothing written to tempdb is intended to be persisted.

What is important to know is that while SQL Server uses the tempdb database for nearly every SQL Server workload, there is only one tempdb per SQL Server instance and tempdb is recreated every time SQL Server is restarted.

##### Tempdb workloads

The main difference between tempdb and other databases is the workload. With tempdb we are constantly creating and destroying objects such as temp tables. This is especially true in heavy OLTP environments where you may have many threads doing all kinds of work and if you are already seeing resource contention on the system the impact will be amplified.

##### So, what is stored in tempdb?

Of course, temp tables and table variables will go into tempdb – this is usually the first object type we think about. So, whenever you create a temporary table in a stored procedure or in a regular batch, that is going to go to tempdb.

Row versions go into tempdb, if you are using snapshot isolation, read committed snapshot where every time a row is modified by a specific transaction, the database engine will store a version of the previously committed image of the row in tempdb.

Hash operations will spill to tempdb. Worktables are also used for spools, cursors, sorts, and temporary large object (LOB) storage – this will all go to tempdb.

Triggers use the row version store in tempdb—this will go to tempdb.

Online index operations—if you are maintaining your indexes with the ONLINE ON keyword then we are creating temporary shadow copies in tempdb.

DBCC CHECKDB also creates shadow copies in tempdb.

As you can see, there’s a lot that goes into tempdb—tempdb will be used for user objects such as global or local temporary tables and indexes, stored procedures, table variables, table-valued functions, and cursors. But we also use tempdb for internal scenarios—such as spills to disk and as worktables that store intermediate results for spools and sorts.

##### What causes contention in tempdb?

Since tempdb is used for so many different scenarios, there is only one tempdb database per SQL Server instance, and we have started pushing towards bigger machines with larger workloads, we have started seeing concurrency issues emerge in the tempdb space in three key areas:

Object allocation contention

Metadata contention

Temp table cache contention

##### What is object allocation contention?

Again, tempdb is structured just like any other database, but remember—the workloads are different in tempdb, so object allocation contention matters more because of the constant creation and destruction of objects.

On a server that is experiencing object allocation contention, you may notice severe blocking especially when the server is experiencing a heavy load. As a result, SQL Server workloads will be slowed, but the server’s CPU may appear to be underutilized. This is because the contention resides in the system metadata. To help avoid these areas of contention, several long-standing best practices have been recommended by SQL Server support teams.

For tempdb, one of the key best practices has always been to create multiple primary data files (mdf) at the same size and same growth rate. The reason for this was to help alleviate the object allocation contention by distributing tempdb activity across multiple partitions.

SQL Server databases must have a primary data file that uses the .mdf file extension by default and a log data file that uses the .ldf file extension. The primary tempdb data file, tempdb.mdf has key pages that track how objects are allocated in SQL Server. As we see in the image below, the table under the tempdb.mdf title represents pages.

A picture containing text, screenshot, number, font

Description automatically generated

Page types used in the data files of the tempdb database.

Page 0 is a header page, and this is true for any primary or secondary data files in SQL Server.

Page 1 is what is called a page free space (PFS) page which is used any time SQL needs to allocate space to an object. Basically, the PFS page contains info on how full the pages are, for the next 8088 pages, in the database. If SQL Server needs to add some data, SQL Server uses the PFS page to see how full the associated object is to see where the data can fit.

After 8088 pages, there is another PFS page in the same data file – it repeats itself. So, you will have more than one PFS page depending on how large the file is.

Page 2 is always the global allocation map (GAM), and this is where the extent allocation comes in as the GAM tracks when SQL Server needs to allocate a uniform extent.

An extent in SQL Server is comprised of 8 x 8KB pages, 64KB, and this is usually the unit of data allocation so if you have a table that’s larger than eight pages, any time we allocate space to that table we will create a full extent, and this is a uniform extent because all eight pages in that extent belong to that object.

So, any time SQL needs to allocate a uniform extent to an object SQL will go to the GAM page and check its availability. The GAM is a bitmap, so if the bit is 1 then that extent is available to be allocated, and if it is 0 then it is not available to be allocated. Once SQL Server has allocated the extent, it just flips the bit for that GAM page from 1 to 0 to show that it is no longer available.

The larger the file, the more GAM Pages you will have, for the GAM you will get another GAM page after 63,904 extents.

Page 3 is for the shared global allocation map (SGAM) and this page is used if SQL needs to allocate space to a mixed extent. This SGAM page tracks mixed extent usage for when an extent is being used by more than one object. So, if I have an object that is less than eight pages and I don’t want to allocate to a full extent, we will use a mixed extent. By default, when a brand-new object is created the first eight pages will be allocated to a mixed extent.

The SGAM is a bitmap, so if the bit is 1 then it’s being used as a mixed extent and has space available to be allocated, then we would look for the corresponding PFS page to find the empty pages within that extent, and then we would allocate that page. The important point here is the SGAM is used in conjunction with the PFS page to allocate space to a mixed extent and again after about 64,000 extents you get another SGAM on the same file.

When you have multiple files in tempdb, you get another immediate header, PFS, GAM, and SGAM because all files will have the same structure and with multiple files we try and share the workload over these files.

SQL Server spreads object allocations across files in the same filegroup based on the proportional fill algorithm. We try to keep the same percentage of free space in each file within the file group so if all the files in the file group start at the same size and they stay the same size and they have the same amount of free space then we turn that proportional fill algorithm into a round robin algorithm—each subsequent allocation will hit the next file and so on—and that was the reason we recommend having multiple files at the same size—spreading object allocations across all the files allows you to get around this object allocation bottleneck. That recommendation came out in SQL Server 2000, and it is still true in SQL Server 2019.

Multiple files equally sized is our best practice and this will stand until testing proves otherwise.

##### Tracking object allocation contention

Prior to SQL Server 2019, the best approach was to monitor the sys.dm\_os\_waiting\_tasks dynamic management view and log the contention history over time.

Consider the SQL Server statement listed below:

A picture containing text, font, screenshot, algebra

Description automatically generated

When looking at the wait resource, you can monitor the contention knowing that the first number refers to the database, the second number is the file id, and the last number is the page type.

This means that contention on wait resource 2:7:2 is tempdb contention as tempdb database is always database id 2, there is contention on file id #7 with GAM contention as the figure illustrates (page #1 is the PFS, #2 is the GAM, and #3 is SGAM).

These wait resource references are commonly in the format 2:1:1, 2:1:3, and so on.

Any results found on database id 2 indicate that there are requests waiting for tempdb resources and the accumulation of these requests can help database administrators narrow down the root cause of the contention.

In SQL Server 2019 we created new functions to improve tempdb troubleshooting. The sys. fn\_PageResCracker dynamic management function returns the db\_id, file\_id, and page\_id for the given page resource value and sys.dm\_db\_page\_info dynamic management function returns page information like page\_id, file\_id, index\_id, object\_id, and more that are present in a page header. This information is useful for troubleshooting and debugging various performance (lock and latch contention) and corruption issues.

The example query below can be used to better resolve wait resource information for any SQL Server release post SQL Server 2019:

A picture containing text, font, screenshot, document

Description automatically generated

##### What is metadata contention?

The other main type of contention is called metadata contention. This type of contention is not about I/O. This contention occurs in memory when multiple threads are trying to modify the same page in memory at the same time.

You can track metadata contention using the same methods you would use to track object allocation contention, the difference is instead of the wait resource being 2:1:1, 2:1:2, 2:1:3 on the PFS, GAM, and SGAM, you are more likely to see the contention occurring on index and data pages and the page number in the wait resource will be a higher value such as 2:1:111, 2:1:118, or 2:1:122, for example.

For metadata content, it is useful to make note of page numbers greater than single digits, track the object name, and the page type description. The object names will show as system tables such as sysallocunits, syscolpars, sysjobactivity, sysscalartypes, sysschobjs, and so on.

Metadata contention was addressed in SQL Server 2019 with the memory-optimized tempdb metadata improvement.

Memory-optimized metadata tables for tempdb is basically a combination of the in-memory OLTP and the temp table metadata features. We took the system tables and the tempdb system tables—and we moved those into non-durable memory-optimized tables.

Remember, that tempdb is temporary—it gets dropped and recreated every time you restart SQL Server so there was no reason for the metadata to be durable. We converted the 12 system tables that are involved in object tracking and tempdb into memory optimized non-durable tables.

We don’t need a specialized memory-optimized file group for tempdb since it is non-durable anyway. All of it is “in memory”—no disk is needed and with memory-optimized tables there’s no latching and no locking. We can massively increase the concurrency against these metadata tables using these lock-free, latch-free data structures.

Enabling memory-optimized metadata tables does require a restart since we must configure the Hekaton DLLs. This was a big improvement in SQL Server 2019 that will eliminate a lot of the metadata contention.

##### Temp table cache contention

In past SQL Server releases starting with SQL Server 2005, we introduced temp table caching to get around some of the metadata caching contention. Basically, when you cache a temp table object—when you delete that table SQL Server doesn’t drop the metadata—we keep a cache of all the temporary objects that are used through a stored procedure and then we reuse the metadata for those objects when you call the same stored procedure with the same temp table again.

As a result, temp table caching has fewer hits to the metadata and alleviates some of that metadata contention—but not completely.

Temp table caching helped address metadata contention by allowing us to reuse tables that didn’t change between stored procedure executions. If the table was not altered after it was created, it would be eligible to be reused by another execution of the same stored procedure. However, if the table is altered (by adding an index or a column, for example), then it can’t be reused and must be dropped when the stored procedure completes.

There are several different tables that we need to delete metadata from to completely drop the table, and this was all being done synchronously at the end of the stored procedure execution. Additionally, every time a new feature is added to SQL Server (ColumnStore indexes, temporal tables, In Memory OLTP, etc.) all these new features require new metadata to be tracked, therefore the number of system tables we need to delete from is increasing, which makes the process more impactful.

Temp table cache contention can be more prominent on larger SQL Server environments, larger core counts and as the size of the cache and the number of concurrent threads accessing the cache grows, this can introduce slower cache access as well as contention for the memory object associated with the cache.

This condition can manifest in two different ways: CMEMTHREAD waits and SOS\_CACHESTORE spinlock waits. To address temp table cache contention, it is recommended to track these wait conditions for evidence and ensure you have installed the latest cumulative updates (CUs) for SQL Server.

### SQL Server 2022 tempdb improvements

In SQL Server 2022 we have addressed the final common areas of contention by introducing concurrent GAM and SGAM updates like the Concurrent PFS updates.

We use the Global Allocation Map (GAM) pages when we are looking for uniform extents and the Shared Global Allocation Map (SGAM) pages when we are looking for mixed extents in tempdb.

In previous releases, under higher concurrent workloads we may have GAM contention where many different threads attempt to allocate extents on the same GAM page and each thread must first wait for another thread to release their UPDATE latch before they can obtain their own latch to allow them to make changes—so, we are just waiting in line.

As can be seen in the workload example below, on SQL Server 2019 there is a wall of GAM contention driving over 123,000 counts of contention with the longest wait taking 949 milliseconds.

A screenshot of a graph

Description automatically generated with low confidence

Dashboard showing the performance of SQL Server 2019 and 123,000 latch contentions over the last five minutes.

The reason for this is that with the update latch, only one thread can modify the GAM page at a time, leading to contention. This is the primary reason we still need multiple data files and because of this contention, SQL Server throughput is decreased and workloads that require many updates to the GAM page will take longer to complete while the machine’s CPU will be underutilized. This contention is due to the workload volume and especially the use of repetitive creation-and-drop operations.

Starting with SQL Server 2016, we changed the default behavior to always allocate uniform extents when creating new objects in tempdb. This helped avoid most of the SGAM contention, but we still use mixed extents for Index Allocation Map (IAM) page allocations. IAM pages are used to track all the pages that belong to an object, so every object that gets created has at least one IAM page. For most workloads, these IAM page allocations don’t cause any issues, but for extremely busy tempdb workloads with many threads of concurrent allocations, these IAM page allocations can still cause SGAM contention.

### SQL Server 2022 addresses GAM and SGAM contention

SQL Server tempdb contention is nearly completely addressed in SQL Server 2022 and these benefits are on by default. With these improvements in SQL Server 2022 we allow concurrent updates to the GAM and SGAM under a shared latch rather than using the update latch. This improvement erases nearly all tempdb contention allowing parallel threads to be able to modify the GAM and SGAM pages as can be seen in the example below.

A screenshot of a computer

Description automatically generated with low confidence

Dashboard showing a similar workload run on SQL Server 2022 with only 607 points of contention over the same period.

In the SQL Server 2022 workload example shown here, we only have 607 points of contention over the same time compared to SQL Server 2019 with the longest wait at only 342 milliseconds. The only contention in the environment was metadata contention in this example because we did not enable the SQL Server optimized tempdb metadata improvement.

There are still possible points of metadata contention, but with SQL Server 2022, the points of contention will be rare and should not lead to any significant performance challenges.

If concurrent GAM and concurrent SGAM updates are some of the last areas of contention, do we still need the best practices to maintain multiple data files for tempdb?

Out of the gate, we are going to continue recommending the same best practices, but we may adjust if we find that it is no longer required through customer feedback.

### System page latch concurrency enhancements in SQL Server 2022

Follow these steps to demonstrate system page latch concurrency enhancements for SQL Server 2022

**Prerequisites**

* VM with at least 8 CPUs and 24Gb RAM
* SQL Server 2022 CTP 2.0
* During setup configure 1 tempdb data file at 50Gb and 1 tempdb log file at 120Gb
* SQL Server Management Studio (SSMS) Version 19 Preview
* Download **ostress.exe** from <https://www.microsoft.com/en-us/download/details.aspx?id=103126>.

**Set up the demo.**

1. Configure perfmon to track batch requests/sec and Page latch waits/Waits started per second.

**Observe performance of a tempdb based workload without metadata optimization and without new SQL Server 2022 enhancements.**

1. Run **disableoptimizetempdb.cmd** and then **disablegamsgam.cmd** from the command prompt.

**Note**: This will ensure tempdb metadata optimization is OFF and turn on two trace flags to disable GAM/SGAM concurrency enhancements. These trace flags are not documented and not supported for production use. They are only used to demonstrate new built-in enhancements.

1. Run **tempsql22stress.cmd 100** from the command prompt.
2. Observe perfmon stats.
3. Execute **pageinfo.sql** from SSMS and observe that all the latch waits are for system table page latches.
4. Observe final duration elapsed from **tempsql22stress.cmd.**

**Observe performance with tempdb metadata optimization enabled but without new SQL Server 2022 enhancements.**

1. Run **optimizetempdb.cmd.**
2. Run **disablegamsgam.cmd.**
3. Run **tempsql22stress.cmd 100.**
4. Observe perfmon stats. Performance is worse even with tempdb metadata optimization enabled.
5. Execute **pageinfo.sql** from SSMS and observe that all the latch waits are for GAM pages.
6. Observe final duration elapsed from **tempsql22stress.cmd**. It is worse than without any optimization.

**Observe performance with tempdb metadata optimization enabled and with new SQL Server 2022 enhancements.**

You could set up SQL Server with only one tempdb data file so one thing you could do is add more files. However, SQL Server 2022 includes enhancements to avoid latch contention for GAM and SGAM pages.

1. Restart SQL Server

Tempdb metadata optimization is already enabled and by restarting you are no longer using trace flags to disable new SQL Server 2022 enhancements.

1. Run **tempsql22stress.cmd 100.**
2. Observe perfmon stats. Performance has greatly increased with no observable latch waits.
3. Execute pageinfo.sql from SSMS and observe there are no latch waits.
4. Observe final duration elapsed from **tempsql22stress.cmd**. It is significantly faster than all other tests.

You have now achieved maximum performance with tempdb workloads and did not have to do any special configuration for tempdb files.

Note: This demo showed that you no longer may have to create multiple tempdb files to avoid system page latch contention. However, it is recommended to use the default setting from SQL Server setup for the number of files.

1. Run **optimizetempdb.cmd.**
2. Run **tempsql22stress.cmd 100.**
3. Observe perfmon stats and final duration. No latch waits and the best performance.

### Additional readings

* [Improve scalability with system page latch concurrency enhancements in SQL Server 2022 - Microsoft SQL Server Blog](https://cloudblogs.microsoft.com/sqlserver/2022/07/21/improve-scalability-with-system-page-latch-concurrency-enhancements-in-sql-server-2022/)

## Query Store on secondary replicas

The Query Store for secondary replicas feature enables the same Query Store functionality on secondary replica workloads that are available for primary replicas. When Query Store for secondary replicas is enabled, replicas send the query execution information that would normally be stored in the Query Store back to the primary replica. The primary replica then persists with the data to disk within its own Query Store. In essence, there is one Query Store shared between the primary and all secondary replicas. The Query Store exists on the primary replica and stores data for all replicas together. Currently, the Query Store for secondary replicas is available with SQL Server 2022 (16.x) instances configured in availability groups. Query Store on secondary replicas enables the same Query Store functionality on secondary replica workloads that are available for primary replicas.

### Enable the Query Store for secondary replicas.

Before using Query Store for secondary replicas on a SQL Server instance, you need to have an [Always On availability group](https://learn.microsoft.com/en-us/sql/database-engine/availability-groups/windows/overview-of-always-on-availability-groups-sql-server?view=sql-server-ver16). Then, enable Query Store for secondary replicas using [ALTER DATABASE SET options (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql-set-options?view=sql-server-ver16).

If the Query Store is not already enabled and in READ\_WRITE mode on the primary replica, you must enable it before proceeding. Execute the following for each desired database on the primary replica:

ALTER DATABASE [Database\_Name] SET QUERY\_STORE = ON;

GO

ALTER DATABASE [Database\_Name] SET QUERY\_STORE

( OPERATION\_MODE = READ\_WRITE );

To enable the Query Store to use all secondary replicas, connect to the primary replica and execute the following for each desired database. Currently, when the Query Store for secondary replicas is enabled, it is enabled for all secondary replicas.

ALTER DATABASE [Database\_Name]

FOR SECONDARY SET QUERY\_STORE = ON (OPERATION\_MODE = READ\_WRITE );

GO

To disable the Query Store on all secondary replicas, connect to the primary replica and execute the following for each desired database:

ALTER DATABASE [Database\_Name]

FOR SECONDARY SET QUERY\_STORE = OFF;

GO

You can evaluate that Query Store is enabled on a secondary replica by connecting to the database on the secondary replica and executing the following:

SELECT desired\_state, desired\_state\_desc, actual\_state, actual\_state\_desc, readonly\_reason

FROM sys.database\_query\_store\_options;

GO

To disable the Query Store for secondary replicas, connect to the database on the primary replicas and run the following code:

ALTER DATABASE CURRENT

FOR SECONDARY SET QUERY\_STORE = OFF;

GO

### Additional readings

* [Query Store for secondary replicas - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/relational-databases/performance/query-store-for-secondary-replicas?view=sql-server-ver16)

## Query Store Hints

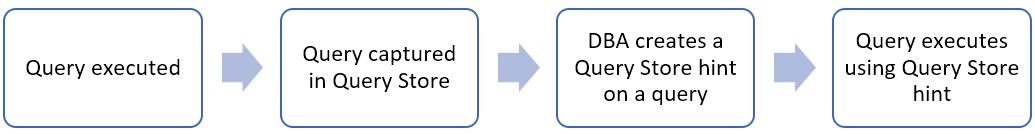
Query Store hints are available in Azure SQL Database and Azure SQL Managed Instance. Query Store hints are also a feature introduced to SQL Server in SQL Server 2022 (16.x). Query Store hints, often referred to as "plan forcing", is a feature in SQL Server that allows DBAs to instruct the SQL Server query processor to use a specific execution plan for a query. This can be a very useful tool when dealing with problematic queries, where the query optimizer isn't selecting the most efficient execution plan.

Query Store hints provide an easy-to-use method for shaping query plans without changing application code.

Ideally the Query Optimizer selects an optimal execution plan for a query. When this doesn't happen, a developer or DBA may wish to manually optimize for specific conditions. Query hints are specified via the OPTION clause and can be used to affect query execution behavior. While query hints help provide localized solutions to various performance-related issues, they do require a rewrite of the original query text. Database administrators and developers may not always be able to make changes directly to Transact-SQL code to inject a query hint. The Transact-SQL may be hard-coded into an application or automatically generated by the application. Previously, a developer may have to rely on [plan guides](https://learn.microsoft.com/en-us/sql/relational-databases/performance/plan-guides?view=sql-server-ver16), which can be complex to use.

### When to use Query Store hints

As the name suggests, this feature extends and depends on the [Query Store](https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver16). Query Store enables the capturing of queries, execution plans, and associated runtime statistics. Introduced in SQL Server 2016 (13.x) and on-by-default in Azure SQL Database, Query Store greatly simplifies the overall performance tuning customer experience.



First the query is executed, then captured by the Query Store. Then the DBA creates a Query Store hint on a query. Thereafter, the query is executed using the Query Store hint.

Examples where Query Store hints can help with query-level performance issues:

* Recompile a query on each execution.
* Cap the memory grant size for a bulk insert operation.
* Limit the maximum degree of parallelism when updating statistics.
* Use a Hash join instead of a Nested Loops join.
* Use [compatibility level](https://learn.microsoft.com/en-us/sql/relational-databases/databases/view-or-change-the-compatibility-level-of-a-database?view=sql-server-ver16) 110 for a specific query while keeping everything else in the database at compatibility level 150.
* Disable row goal optimization for a SELECT TOP query.

To use Query Store hints:

1. Identify the Query Store query\_id of the query statement you wish to modify. You can do this in various ways: 1.1. Querying the [Query Store catalog views](https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/query-store-catalog-views-transact-sql?view=sql-server-ver16). 1.2. Using SQL Server Management Studio built-in Query Store reports. 1.3. Using Azure portal Query Performance Insight for Azure SQL Database.
2. Execute sys.sp\_query\_store\_set\_hints with the query\_id and query hint string you wish to apply to the query. This string can contain one or more query hints. For complete information, see [sys.sp\_query\_store\_set\_hints](https://learn.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sys-sp-query-store-set-hints-transact-sql?view=sql-server-ver16).

Once created, Query Store hints persist and survive restarts and failovers. Query Store hints override hard-coded statement-level hints and existing plan guide hints.

If a query hint contradicts what is possible for query optimization, query execution is not blocked, and the hint isn't applied. In the cases where a hint would cause a query to fail, the hint is ignored and the latest failure details can be viewed in [sys.query\_store\_query\_hints](https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-query-store-query-hints-transact-sql?view=sql-server-ver16).

### Query Store hints and feature interoperability

* Query Store hints will override other hard-coded statement level hints and plan guides.
* Queries will always execute and opposing the Query Store hints are ignored that would otherwise cause an error.
* If Query Store hints contradict, SQL Server does not block query execution, and Query Store hint is not applied.
* Simple parameterization - Query Store hints are not supported by statements that qualify for simple parameterization.
* Forced parameterization - The RECOMPILE hint is not compatible with forced parameterization set at the database level. If the database has a forced parameterization set, and the RECOMPILE hint is part of the hints string set in Query Store for a query, SQL Server will ignore the RECOMPILE hint and will apply any other hints if they are applied.
  + Additionally, SQL Server will issue a warning (error code 12461) stating that the RECOMPILE hint was ignored.
* Manually created Query Store hints are exempt from cleanup. The hint and the query will not be cleaned up from Query Store by the automatic retention of the capture policy.
  + Queries can be [manually removed by users](https://learn.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-query-store-remove-query-transact-sql?view=sql-server-ver16), which would also remove the associated Query Store hint.
  + Query Store hints automatically generated by the [CE Feedback](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing-details?view=sql-server-ver16#cardinality-estimation-ce-feedback) are subject to clean up by the automatic retention of the capture policy.
  + [DOP feedback](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing-feedback?view=sql-server-ver16#degree-of-parallelism-dop-feedback) and [memory grant feedback](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing-details?view=sql-server-ver16#memory-grant-feedback) shape query behavior without using Query Store hints. When queries are cleanup by automatic retention of the capture policy, DOP feedback and memory grant feedback data is also cleaned up.
  + You can manually create the same Query Store hint that CE feedback implemented, and then the query with the hint would no longer be subject to clean up by the automatic retention of the capture policy.

### Query Store hints best practices

* Complete index and statistics maintenance before evaluating queries for potential new Query Store hints.
* Test your application database on the latest [compatibility level](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql-compatibility-level?view=sql-server-ver16) before using Query Store hints.
  + For example, Parameter Sensitive Plan (PSP) optimization was introduced in SQL Server 2022 (16.x) (compatibility level 160), which leverages multiple active plans per query to address non-uniform data distributions. If your environment cannot use the latest compatibility level, Query Store hints using the RECOMPILE hint can be leveraged on any supporting compatibility level.
* Query Store hints override SQL Server query plan behavior. It is recommended to only leverage Query Store hints when it is necessary to address performance-related issues.
* It is recommended to re-evaluate Query Store hints, statement level hints, plan guides, and Query Store forced plans any time data distributions change and during database migrations projects. Changes in data distribution may cause Query Store hints to generate suboptimal execution plans.

**Identify the query in the query store system catalog views:**

SELECT q.query\_id, qt.query\_sql\_text

FROM sys.query\_store\_query\_text qt

INNER JOIN sys.query\_store\_query q ON

qt.query\_text\_id = q.query\_text\_id

WHERE query\_sql\_text like N'%PostalCode =%'

AND query\_sql\_text not like N'%query\_store%';

GO

In the following samples, the previous query example in the SalesLT database was identified as query\_id 39.

**Once identified, apply the hint to enforce a maximum memory grant size in percent of configured memory limit to the query\_id:**

EXEC sys.sp\_query\_store\_set\_hints @query\_id= 39, @query\_hints = N'OPTION(MAX\_GRANT\_PERCENT=10)';

**You can also apply query hints with the following syntax, for example the option to force the**[**legacy cardinality estimator**](https://learn.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server?view=sql-server-ver16)**:**

EXEC sys.sp\_query\_store\_set\_hints @query\_id= 39, @query\_hints = N'OPTION(USE HINT(''FORCE\_LEGACY\_CARDINALITY\_ESTIMATION''))';

**You can apply multiple query hints with a comma-separated list:**

EXEC sys.sp\_query\_store\_set\_hints @query\_id= 39, @query\_hints = N'OPTION(RECOMPILE, MAXDOP 1, USE HINT(''QUERY\_OPTIMIZER\_COMPATIBILITY\_LEVEL\_110''))';

**Review the Query Store hint in place for query\_id 39:**

SELECT query\_hint\_id, query\_id, query\_hint\_text, last\_query\_hint\_failure\_reason, last\_query\_hint\_failure\_reason\_desc, query\_hint\_failure\_count, source, source\_desc

FROM sys.query\_store\_query\_hints

WHERE query\_id = 39;

**Finally, remove the hint from query\_id 39, using**[**sp\_query\_store\_clear\_hints**](https://learn.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sys-sp-query-store-clear-hints-transact-sql?view=sql-server-ver16)**.**

EXEC sys.sp\_query\_store\_clear\_hints @query\_id = 39;

### Additional Readings

* [Query Store hints - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/relational-databases/performance/query-store-hints?view=sql-server-ver16)
* [Query Store hints best practices - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/relational-databases/performance/query-store-hints-best-practices?view=sql-server-ver16)

## Resumable Table Add Constraints

The resumable operation for online index creation and rebuilding are already supported for SQL Server 2019, Azure SQL Database, and Azure SQL Managed Instance. The resumable operations allow index operations to be executed while the table is [online](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-table-transact-sql?view=sql-server-ver16#with--online--on--off-as-applies-to-altering-a-column) (ONLINE=ON) and also:

* Pause and restart an index create or rebuild operation multiple times to fit a maintenance window.
* Recover from index creation or rebuild failures, such as database failovers or running out of disk space.
* Enable truncation of transaction logs during an index creation or rebuild operation.
* When an index operation is paused, both the original index and the newly created one require disk space and need to be updated during [Data Manipulation Language (DML)](https://learn.microsoft.com/en-us/sql/t-sql/statements/statements?view=sql-server-ver16#data-manipulation-language) operations.

The new extensions for SQL Server 2022, SQL Database, and SQL Managed Instance allow a resumable operation for the [Data Definition Language (DDL)](https://learn.microsoft.com/en-us/sql/t-sql/statements/statements?view=sql-server-ver16#data-definition-language) command [ALTER TABLE ADD CONSTRAINT](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-table-transact-sql?view=sql-server-ver16) and adding a Primary or Unique Key. For more information on adding a Primary or Unique Key, see [ALTER TABLE table\_constraint](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-table-table-constraint-transact-sql?view=sql-server-ver16).

### Online/Resumable Index Operations

##### Offline index operations​

Index operations executed offline – default (ONLINE=OFF), lock a table in exclusive mode​

##### Online index operations​

* Increase table availability​.
* Updates can be executed during index operation on the same table​.
* Refer to create/rebuild/drop index online with option ONLINE=ON​

##### Resumable index operation allows to​.

* Pause and restart an index create or rebuild operation multiple times to fit​  
  a maintenance window​.
* Recover from index create or rebuild failures (database failovers, or out of disk space)​
* Enable truncation of transaction logs during an index create or rebuild operation​.
* Supported only with RESUMABLE=ON and ONLINE=ON options ​

### Table and Column Constrains

Used to define rules and conditions affecting data in a table. ​

##### Table and column constraints​

* PRIMARY KEY – uniquely identifies each row in a table​.
* FOREIGN KEY - defines referential integrity between tables​.
* UNIQUE - ensures date uniqueness for a column or group column​.
* NOT NULL - ensures that a column has no NULL values​.
* CHECK - ensures a specific condition for a column​.
* DEFAULT- sets default value for a column if not specified​

### Resumable Table Add Constraints

* Based on “resumable index operations” implemented for SQL indexes​
* Allow to ​
* Pause and restart multiple times a table constraint being built​.
* Recover from failure during table constraint build (database failover, disk space)​
* Enable truncation of transaction logs during add constraints​.
* Resumable operation is created using ALTER TABLE ADD. CONSTRAINT​
* Supported only with RESUMABLE=ON and ONLINE=ON options​

### Resumable Table Add Constraints - Syntax ​

##### ADD primary key constraint​.

   ALTER TABLE table\_name ADD CONSTRAINT PK\_Constraint PRIMARY KEY (col1, col2, ..) ​

         WITH (ONLINE = ON, MAXDOP = N, RESUMABLE = ON, MAX\_DURATION=<time in minutes>) ​

* ADD unique constraint​.

   ALTER TABLE table\_name ADD CONSTRAINT UC\_Constraint UNIQUE [CLUSTERED] (col1, col2,..) ​

        WITH (ONLINE = ON, MAXDOP = N, RESUMABLE = ON, MAX\_DURATION=<time in minutes>) ​

##### Pause add constraint. ​

   ALTER INDEX ALL ON test PAUSE;​

##### Resume, add constraint​.

   ALTER INDEX ALL ON test RESUME​

##### Abort add constraint​.

   ALTER INDEX ALL ON test ABORT;​

### Check Resumable Status and Table Constraints

##### Check the current resumable status during add constraint operation​.

    SELECT \* FROM sys.index\_resumable\_operations;​

​

##### List constraints​

  SELECT constraint\_name, table\_name, constraint\_type ​  
      FROM INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS​  
        WHERE CONSTRAINT\_TYPE='PRIMARY KEY' or CONSTRAINT\_TYPE= 'UNIQUE' ​

​

##### Show constraint information​.

        SELECT \* FROM sys.indexes WHERE object\_id=object\_id('table\_name'); ​

### Additional reading

* [SQLEAP/docs/resumable-constraints at main · microsoft/SQLEAP (github.com)](https://github.com/microsoft/SQLEAP/tree/main/docs/resumable-constraints)​
* [Guidelines for Online Index Operations - SQL Server | Microsoft Docs](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/guidelines-for-online-index-operations?view=sql-server-ver15)​
* [Perform Index Operations Online - SQL Server | Microsoft Docs](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/perform-index-operations-online?view=sql-server-ver15)​
* [How Online Index Operations Work - SQL Server | Microsoft Docs](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/how-online-index-operations-work?view=sql-server-ver15)​
* [Resumable add table constraints - SQL Server & Azure SQL | Microsoft Learn](https://learn.microsoft.com/en-us/sql/relational-databases/security/resumable-add-table-constraints?view=sql-server-ver16)

# Security:

## Always Encrypted with Secure Enclaves

### Protecting Data Through its Lifecycle

Protective sensitive business data is more important today than ever before. Corporate trade secrets, national security information, personal medical records, Social Security, and credit card numbers are all stored, used, and transmitted online and through connected devices.

Data needs to be protected in three states: at rest, in use, and in motion. Each state presents unique security challenges.

### Data at Rest

Data is at rest when it is stored on a hard drive. In SQL Server/Azure SQL Database, this means data in database files (.mdf), log files (.ldf), backup files (.bak), bacpac, etc. Either encrypting the entire drive or encrypting files by using Transparent Database Encryption (TDE), is one of the best ways to ensure the security of data at rest.

TDE ensures encryption of database files at the page level. The pages in an encrypted database are encrypted before they are written to disk and decrypted when read into memory.

### Data in transit

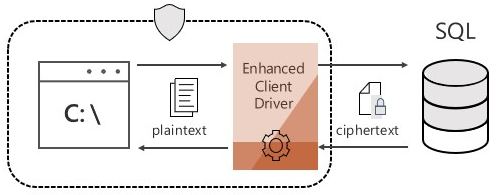
Data is in transit when it is being transmitted between components or tiers of a computer system, either over the public or untrusted network. Specifically, in the database world: when it is being transmitted between a database and a database client. Transport Layer Security (TLS) is a mechanism that is commonly used to protect data in transit.

### Data in Use

Data in use is data that is stored in a non-persistent memory, CPU caches or registers – in other words, the data used during computations. Most database systems support both encryption at rest and in transit. Always Encrypted encrypts data while in use.

### Always Encrypted

Always Encrypted protects sensitive data in use from high-privileged yet unauthorized SQL users both on-premises and in the cloud.



* Client-side Encryption - Client-side encryption of sensitive data using keys   
  that are never given to the database system.
* Encryption Transparency - Client driver transparently encrypts query parameters and decrypts encrypted results.
* Queries on Encrypted Data - Support for equality comparison, including joins, group by and distinct operators via deterministic encryption.

The Database Engine stores encryption configuration for each column in database metadata. Note, however, the Database Engine never stores or uses the keys in plaintext. It only stores encrypted values of column encryption keys and information about the location of column master keys, which are stored in external trusted key stores, such as Azure Key Vault, Windows Certificate Store on a client machine, or a hardware security module.

For more information, see [Always Encrypted Cryptograph | SQL Docs](https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-cryptography?view=sql-server-ver15).

Introduced in SQL Server 2016, Always Encrypted protects the confidentiality of sensitive data from malware and high-privileged unauthorized users of SQL Server. High-privileged unauthorized users are DBAs, computer admins, cloud admins, or anyone else who has legitimate access to server instances, hardware, etc., but who should not have access to some or all the actual data.

Always Encrypted protects the data by encrypting it on the client side and never allowing the data or the corresponding cryptographic keys to appear in plaintext inside the SQL Server Engine. As a result, the functionality on encrypted columns inside the database is severely restricted. The only operations SQL Server can perform on encrypted data are equality comparisons (and only available on columns with deterministic encryption). All other operations, including cryptographic operations (initial data encryption or key rotation) and or rich computations (for example, pattern matching) are not supported inside the database. Users need to move their data outside of the database to perform these operations on the client-side.

**Always Encrypted Challenges**

* Many applications require support for queries that do more than equality comparison.
* Data needs to be moved out of the database for initial encryption and key rotation.

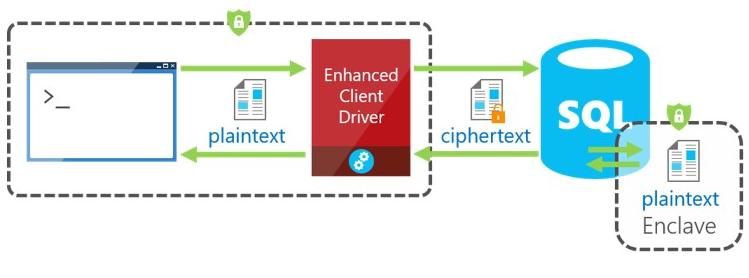
To address these challenges Always Encrypted with secure enclaves expands confidential computing capabilities of Always Encrypted by enabling in-place encryption and richer confidential queries. Always Encrypted with secure enclaves is available in SQL Server 2019 (15.x), sql server 2022, and in Azure SQL Database.

### Always Encrypted with Secure Enclaves

Always Encrypted *with secure enclaves* addresses these limitations by

* allowing computations on plaintext data inside a secure enclave on the server side.
* A secure enclave is a protected region of memory within the SQL Server process and acts as a trusted execution environment for processing sensitive data inside the SQL Server engine. A secure enclave appears as an opaque box to the rest of the SQL Server and other processes on the hosting machine.
* There is no way to view any data or code inside the enclave from the outside, even with a debugger.
* For In-place encryption - cryptographic operations on sensitive data, for example: initial data encryption or rotating a column encryption key, are performed inside the secure enclave and do not require moving the data outside of the database.

SQL server 2022, now has Support for JOIN, GROUP BY, and ORDER BY, and for text columns using UTF-8 collations in confidential queries using enclaves. Improved performance. Can perform rich computation - operations on encrypted columns, including pattern matching (the LIKE predicate) and range comparisons, are supported inside the secure enclave, which unlocks Always Encrypted to a broad range of applications and scenarios that require such computations to be performed inside the database system.



When parsing an application's query, the SQL Server Engine determines if the query contains any operations on encrypted data that require the use of the secure enclave. For queries where the secure enclave needs to be accessed:

* The client driver sends the column encryption keys required for the operations to the secure enclave (over a secure channel).
* Then, the client driver submits the query for execution along with the encrypted query parameters.

During query processing, the data or the keys are not exposed in plain text in the SQL Server Engine, only in the secure enclave. The secure enclave decrypts the parameters and the data stored in encrypted columns and performs the requested operations.

For more information:

* Tutorial: <https://aka.ms/AlwaysEncryptedEnclavesTutorial>
* Documentation at <https://aka.ms/AlwaysEncryptedwithSecureEnclaves>
* Blog: <https://blogs.msdn.microsoft.com/sqlsecurity/tag/always-encrypted/>
* GitHub: <https://github.com/microsoft/sql-server-samples/tree/master/samples/features/security/always-encrypted-with-secure-enclaves>

## SQL Ledger

Establishing trust around the integrity of data stored in database systems has been a longstanding problem for all organizations that manage financial, medical, or other sensitive data. The ledger feature provides tamper-evidence capabilities in your database. You can cryptographically attest to other parties, such as auditors or other business parties, that your data hasn't been tampered with.

Ledger helps protect data from any attacker or high-privileged user, including database administrators (DBAs), system administrators, and cloud administrators. As with a traditional ledger, the feature preserves historical data. If a row is updated in the database, its previous value is maintained and protected in a history table. Ledger provides a chronicle of all changes made to the database over time.

Ledger and the historical data are managed transparently, offering protection without any application changes. The feature maintains historical data in a relational form to support SQL queries for auditing, forensics, and other purposes. It provides guarantees of cryptographic data integrity while maintaining the power, flexibility, and performance of the SQL database.

database.

A diagram of a server

Description automatically generated with low confidence

##### Use cases for ledger.

Let's go over some advantages for using ledger.

**Streamlining audits**

Any production system's value is based on the ability to trust the data that the system is consuming and producing. If a malicious user has tampered with the data in your database, that can have disastrous results in the business processes relying on that data.

Maintaining trust in your data requires a combination of enabling proper security controls to reduce potential attacks, backup and restore practices, and thorough disaster recovery procedures. Audits by external parties ensure that these practices are put in place.

Audit processes are highly time-intensive activities. Auditing requires on-site inspection of implemented practices such as reviewing audit logs, inspecting authentication, and inspecting access controls. Although these manual processes can expose potential gaps in security, they can't provide attestable proof that the data hasn't been maliciously altered.

Ledger provides cryptographic proof of data integrity to auditors. This proof can help streamline the auditing process. It also provides nonrepudiation regarding the integrity of the system's data.

**Multiple-party business processes**

In some systems, such as supply-chain management systems, multiple organizations must share state from a business process with one another. These systems struggle with the challenge of sharing and trusting data. Many organizations are turning to traditional blockchains, such as Ethereum or Hyperledger Fabric, to digitally transform their multiple-party business processes.

Blockchain is a great solution for multiple-party networks where trust is low between parties that participate on the network. Many of these networks are fundamentally centralized solutions where trust is important, but a fully decentralized infrastructure is a heavyweight solution.

Ledger provides a solution for these networks. Participants can verify the integrity of the centrally housed data, without the complexity and performance implications that network consensus introduces in a blockchain network.

**Trusted off-chain storage for blockchain.**

When a blockchain network is necessary for a multiple-party business process, the ability to query the data on the blockchain without sacrificing performance is a challenge.

Typical patterns for solving this problem involve replicating data from the blockchain to an off-chain store, such as a database. But after the data is replicated to the database from the blockchain, the data integrity guarantees that a blockchain offer is lost. Ledger provides data integrity for off-chain storage of blockchain networks, which helps ensure complete data trust through the entire system.

### How it works

Any rows modified by a transaction in a ledger table is cryptographically SHA-256 hashed using a Merkle tree data structure that creates a root hash representing all rows in the transaction. The transactions that the database processes are then also SHA-256 hashed together through a Merkle tree data structure. The result is a root hash that forms a block. The block is then SHA-256 hashed through the root hash of the block, along with the root hash of the previous block as input to the hash function. That hashing forms a blockchain.

The root hashes in the [database ledger](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-database-ledger?view=sql-server-ver16), also called [Database digests](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-overview?view=sql-server-ver16#database-digests), contain the cryptographically hashed transactions and represent the state of the database. They can be periodically generated and stored outside the database in tamper-proof storage, such as [Azure Blob Storage configured with immutability policies](https://learn.microsoft.com/en-us/azure/storage/blobs/immutable-storage-overview), [Azure Confidential Ledger](https://learn.microsoft.com/en-us/azure/confidential-ledger/index) or on-premises [Write Once Read Many (WORM) storage devices](https://en.wikipedia.org/wiki/Write_once_read_many). Database digests are later used to verify the integrity of the database by comparing the value of the hash in the digest against the calculated hashes in database.

Ledger functionality is introduced to tables in two forms:

* [Updatable ledger tables](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-overview?view=sql-server-ver16#updatable-ledger-tables), which allow you to update and delete rows in your tables.
* [Append-only ledger tables](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-overview?view=sql-server-ver16#append-only-ledger-tables), which only allow insertions to your tables.

Both updatable ledger tables and append-only ledger tables provide tamper-evidence and digital forensics capabilities.

##### Updatable ledger tables

[Updatable ledger tables](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-updatable-ledger-tables?view=sql-server-ver16) are ideal for application patterns that expect to issue updates and deletions to tables in your database, such as system of record (SOR) applications. Existing data patterns for your application don't need to change to enable ledger functionality.

Updatable ledger tables track the history of changes to any rows in your database when transactions that perform updates or deletions occur. An updatable ledger table is a system-versioned table that contains a reference to another table with a mirrored schema.

The other table is called the *history table*. The system uses this table to automatically store the previous version of the row each time a row in the ledger table is updated or deleted. The history table is automatically created when you create an updatable ledger table.

The values in the updatable ledger table and its corresponding history table provide a chronicle of the values of your database over time. A system-generated ledger view joins the updatable ledger table and the history table so that you can easily query this chronicle of your database.

For more information on updatable ledger tables, see [Create and use updatable ledger tables](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-how-to-updatable-ledger-tables?view=sql-server-ver16).

##### Append-only ledger tables.

[Append-only ledger tables](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-append-only-ledger-tables?view=sql-server-ver16) are ideal for application patterns that are insert-only, such as security information and event management (SIEM) applications. Append-only ledger tables block updates and deletions at the API level. This blocking provides more tampering protection from privileged users such as system administrators and DBAs.

Because only insertions are allowed into the system, append-only ledger tables don't have a corresponding history table because there's no history to capture. As with updatable ledger tables, a ledger view provides insights into the transaction that inserted rows into the append-only table, and the user that performed the insertion.

For more information on append-only ledger tables, see [Create and use append-only ledger tables](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-how-to-append-only-ledger-tables?view=sql-server-ver16).

##### Ledger database

Ledger databases provide an easy solution for applications that require the integrity of all data to be protected for the entire lifetime of the database. A ledger database can only contain ledger tables. Creating regular tables (that are not ledger tables) is not supported. Each table is, by default, created as an [Updatable ledger table](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-updatable-ledger-tables?view=sql-server-ver16) with default settings, which makes creating such tables very easy. You configure a database as a ledger database at creation. Once created, a ledger database cannot be converted to a regular database. For more information, see [Configure a ledger database](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-how-to-configure-ledger-database?view=sql-server-ver16).

##### Database digests

The hash of the latest block in the database ledger is called the [database digest](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-digest-management?view=sql-server-ver16). It represents the state of all ledger tables in the database at the time that the block was generated.

When a block is formed, its associated database digest is published and stored outside the database in tamper-proof storage. Because database digests represent the state of the database at the time that they were generated, protecting the digests from tampering is paramount. An attacker who has access to modify the digests would be able to:

1. Tamper with the data in the database.
2. Generate the hashes that represent the database with those changes.
3. Modify the digests to represent the updated hash of the transactions in the block.

Ledger provides the ability to automatically generate and store the database digests in [immutable storage](https://learn.microsoft.com/en-us/azure/storage/blobs/immutable-storage-overview) or [Azure Confidential Ledger](https://learn.microsoft.com/en-us/azure/confidential-ledger/index), to prevent tampering. Alternatively, users can manually generate database digests and store them in the location of their choice. Database digests are used for verifying that the data stored in ledger tables hasn't been tampered with.

##### Ledger verification

The ledger feature doesn't allow modification of the content of ledger system views, append-only tables, and history tables. However, an attacker or system administrator who has control of the machine can bypass all system checks and directly tamper with the data. For example, an attacker or system administrator can edit the database files in storage. Ledger can't prevent such attacks but guarantees that any tampering will be detected when the ledger data is verified.

The [ledger verification](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-database-verification?view=sql-server-ver16) process takes as input one or more previously generated database digests and recomputes the hashes stored in the database ledger based on the current state of the ledger tables. If the computed hashes don't match the input digests, the verification fails, indicating that the data has been tampered with. Ledger then reports all inconsistencies that it has detected.

### Create an append-only ledger table.

We'll create a KeycardEvents table with the following schema.

| **Column name** | **Data type** | **Description** |
| --- | --- | --- |
| EmployeeID | int | The unique ID of the employee accessing the building |
| AccessOperationDescription | nvarchar (MAX) | The access operation of the employee |
| Timestamp | datetime2 | The date and time the employee accessed the building |

CREATE SCHEMA [AccessControl];

GO

CREATE TABLE [AccessControl].[KeyCardEvents]

(

[EmployeeID] INT NOT NULL,

[AccessOperationDescription] NVARCHAR (1024) NOT NULL,

[Timestamp] Datetime2 NOT NULL

)

WITH (LEDGER = ON (APPEND\_ONLY = ON));

1. Add a new building access event in the [AccessControl]. [KeyCardEvents] table with the following values.

INSERT INTO [AccessControl].[KeyCardEvents]

VALUES ('43869', 'Building42', '2020-05-02T19:58:47.1234567');

1. View the contents of your KeyCardEvents table, and specify the [GENERATED ALWAYS](https://learn.microsoft.com/en-us/sql/t-sql/statements/create-table-transact-sql?view=sql-server-ver16#generate-always-columns) columns that are added to your [append-only ledger table](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-append-only-ledger-tables?view=sql-server-ver16).

SELECT \*

,[ledger\_start\_transaction\_id]

,[ledger\_start\_sequence\_number]

FROM [AccessControl].[KeyCardEvents];

Screenshot that shows results from querying the KeyCardEvents table.

1. View the contents of your KeyCardEvents ledger view along with the ledger transactions system view to identify who added records into the table.

SELECT

t.[commit\_time] AS [CommitTime]

, t.[principal\_name] AS [UserName]

, l.[EmployeeID]

, l.[AccessOperationDescription]

, l.[Timestamp]

, l.[ledger\_operation\_type\_desc] AS Operation

FROM [AccessControl].[KeyCardEvents\_Ledger] l

JOIN sys.database\_ledger\_transactions t

ON t.transaction\_id = l.ledger\_transaction\_id

ORDER BY t.commit\_time DESC;

1. Try to update the KeyCardEvents table by changing the EmployeeID from 43869 to 34184.

UPDATE [AccessControl].[KeyCardEvents] SET [EmployeeID] = 34184;

## granular permissions

In the last 3 years, the SQL Security team has put more emphasis on enabling customers to use SQL Server while adhering to the [Principle of least Privilege (PolP)](https://techcommunity.microsoft.com/t5/azure-sql-blog/security-the-principle-of-least-privilege-polp/ba-p/2067390). As part of that effort, all new features in the next SQL Server release: SQL Server 2022, can be controlled with more granular permissions. SQL Ledger is a good example of such a new feature. It comes with separate permissions for creating (ENABLE LEDGER) vs dropping ledger tables (ALTER LEDGER); a separate permission to view ledger content (VIEW LEDGER CONTENT) for auditing purposes; and another independent permission to generate a ledger digest (GENERATE LEDGER DIGEST).

Besides new features, a lot of effort also went into improving the existing set of permissions, namely by making them more granular. This has happened in 3 distinguishable areas:

### What is new?

**(1) Access to System Metadata**: 10 new permissions (5 on Server- and 5 on Database level)

**(2) Extended Events**: 18 new permissions (9 on Server- and 9 on Database level)

**(3) Security-related objects**: 4 new permissions (3 on Server- and 1 on Database level)

##### Access to System Metadata

System Metadata in SQL Server is exposed mainly via Catalog views, Dynamic Management Views (DMVs) and Functions, as well as a few DBCC commands and extended stored procedures.

Until SQL Server 2019, the permission requirement for Catalog Views was VIEW ANY DEFINITION respectively VIEW DEFINITION depending on the metadata’s origin: Server level or per individual database.

Similarly for DMVs the minimal required permission was VIEW SERVER STATE respectively DATABASE STATE, depending on the applicable scope of the metadata.

**In other words, one single permission covered roughly 300 Catalog Views and another one permission covered about 290 DMVs and DMFs.**

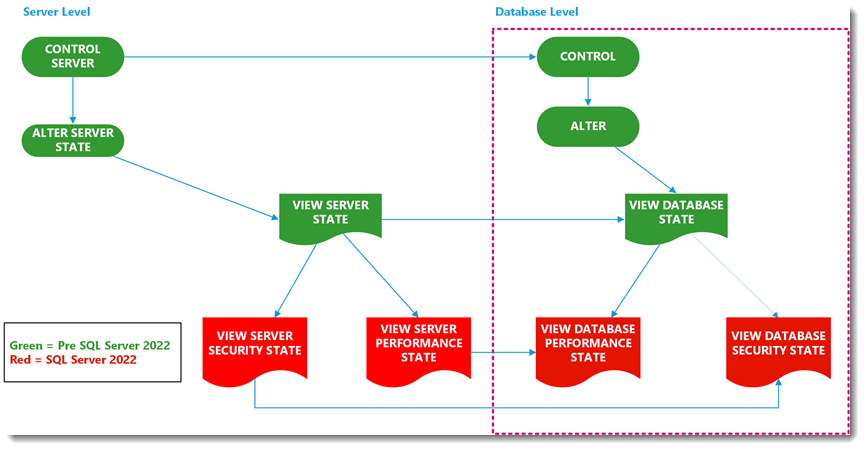
 But system metadata can belong to many different areas. It can relate to performance, high availability features (and those again can be grouped into subareas), fulltextsearch, security, general system information, service broker. And many of those can again be split into subareas such as Availability Groups, Failover Clustering, execution environment, SQL OS, memory, and caches etc.

 However, coming up with individual permissions for each sub-area would not benefit many customers and unnecessarily bloat the permissions system.

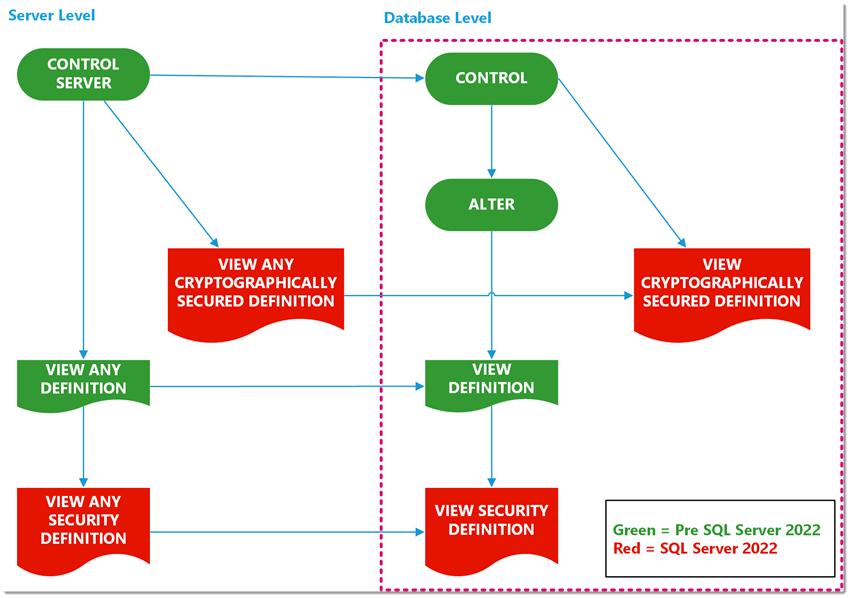
There are many scenarios in which it is desirable to grant certain individuals permissions to fully investigate a system’s behavior but also have confidence, that the person (or app/service account) cannot read security-sensitive information.  
This is often the case with junior admins, external consultants, outsourced database administration services, performance-monitoring tools and similar which mostly need to see performance-related live or statistical information but not information about user accounts, permissions, cryptographic-properties etc..  
While it is usually not a problem if someone who is tasked to analyze memory-usage could also view information about HA-replica states, exposure to security-information is less forgiving.

 Therefore, we split the existing permissions into two separate **sub-permissions** below VIEW SERVER STATE/VIEW DATABASE STATE respectively VIEW ANY DEFINITION/VIEW DEFINITION.

The diagram below depicts this change/additional granularity:

 *Split of view SERVER/DATABASE STATE into VIEW SERVER/DATABASE PERFORMANCE STATE and VIEW SERVER/DATABASE SECURITY STATE*

The same was done for Catalog views, which before this split were covered by a general VIEW DEFINITION:

 *Split of VIEW DEFINITION/ANY DEFINITION into VIEW ANY SECURITY DEFINITION and SECURITY DEFINITION as well as a new permission VIEW CRYPTOGRAPHICALLY SECURED DEFINITION/ANY CRYPTOGRAPHICALLY SECURED DEFINITION*

We are introducing **10 new permissions (5 on server plus 5 on database level)**:

##### Server level:

1. VIEW ANY SECURITY DEFINITION
2. VIEW ANY PERFORMANCE DEFINITION
3. VIEW SERVER SECURITY STATE
4. VIEW SERVER PERFORMANCE STATE
5. VIEW ANY CRYPTOGRAPHICALLY SECURED DEFINITION

##### Database level:

1. VIEW DATABASE SECURITY STATE
2. VIEW DATABASE PERFORMANCE STATE
3. VIEW SECURITY DEFINITION
4. VIEW PERFORMANCE DEFINITION
5. VIEW CRYPTOGRAPHICALLY SECURED DEFINITION

##### What is the difference between VIEW PERFORMANCE STATE and VIEW SECURITY STATE and DEFINITION?

It's best understood by approaching it from the following angle: From all the DMVs/DMFs and Catalog Views, we made a split between those that contain **security-wise sensitive information** and literally “**everything else**”. To put it into numbers: out of approximately 800 system views and functions, about 70 are from now on covered by VIEW SECURITY STATE or SECURITY DEFINITION (depending on whether it’s a DMV/DMF or a Catalog View). Everything else remains covered by one permission: VIEW PERFORMANCE STATE respectively VIEW DEFINITION. (The VIEW (ANY) PERFORMANCE DEFINITION permissions were introduced with SQL Server 2022 RTM.)

**Why “everything else” and what exactly is it?**

Ok, you really want to go down this hole ;)  
Quite simply, everything that is not security-wise sensitive, is covered by “PERFORMANCE”. Even if the content is returning static data, Fulltextsearch wordlists or Cluster Node information. If you look at every DMV in detail you will notice many of them contain a mix of information, crossing subject boundaries. And trying to make up more buckets serves little purpose and would just make life harder for those trying to figure out which permissions to grant with little or no benefit.

 The main scenario that we want to cover: Administrators can grant access to system information without having to worry that the Junior DBA/External Consultant/Remote Admin Service company gains access to information that it could abuse.

And that is achieved by separating out the security-wise sensitive objects.  
When it comes to performance monitoring, troubleshooting and similar tasks, having access to HA-information usually poses no risks for a company. But information about key encryption strengths should not be available without proper cause.

**How the new server roles that were announced recently align to this work.**

In May 2022 I announced 7 new server roles for Azure SQL Database ([New server roles for Azure SQL Database and SQL Server 2022 in Public Preview - Microsoft Tech Commu...](https://techcommunity.microsoft.com/t5/azure-sql-blog/new-server-roles-for-azure-sql-database-and-sql-server-2022-in/ba-p/3428433)). One of these new roles is based on these new permissions:

 ##MS\_SecurityDefinitionReader## => VIEW ANY SECURITY DEFINITION

If you look at the list of new Server roles for SQL Server 2022 you will find even more roles that align 1:1 to these new granular permissions on server level:

##MS\_ServerPerformanceStateReader## => VIEW SERVER PERFORMANCE STATE  
##MS\_ServerSecurityStateReader## => VIEW SERVER SECURITY STATE  
##MS\_PerformanceDefinitionReader## => VIEW ANY PERFORMANCE DEFINITION

[Server-level roles - SQL Server | Microsoft Docs](https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/server-level-roles?view=sql-server-ver16#fixed-server-level-roles-introduced-in-sql-server-2022)

 By using those server roles, these new permissions can be used in Azure SQL Database, Managed Instance and SQL Server alike. This is the only way to delegate server level permissions in Azure SQL Database.

### Extended Events

Extended Events (aka XEvents) are the tracing technology that since SQL Server 2012 replaces SQL Trace & Profiler. ([XEvents overview - SQL Server, Azure SQL Database, and Azure SQL Managed Instance - SQL Server | Mic...](https://docs.microsoft.com/en-us/sql/relational-databases/extended-events/extended-events?view=sql-server-ver15))

Since then, managing XEvent sessions was covered by one, respectively two Permissions: ALTER ANY EVENT SESSION – applicable to either the database level (SQL Database) or server level (SQL Server and MI).

This permission covers everything from creating new sessions, changing any part of the definition like adding or removing attributes or targets and starting and stopping them.

Note on SQL Auditing  
If XEvents are not your main concern, it may help to understand that Security Auditing in SQL is also based on XEvent architecture and therefore shares a lot of the DDL and general workflow.  
While I cannot disclose anything specific, it probably makes sense that the permissions for Auditing will profit from this change that XEvents undergo at some point. ;)

 Starting with SQL Server 2022, XEvent session management can be controlled by **18 additional permissions (9 on server plus 9 on database level).** This allows for much more fine-grained control over what a user is allowed to do with XEvent sessions.

The list of new permissions:

##### Server level:

1. CREATE ANY EVENT SESSION
2. DROP ANY EVENT SESSION
3. ALTER ANY EVENT SESSION OPTION
4. ALTER ANY EVENT SESSION ADD EVENT
5. ALTER ANY EVENT SESSION DROP EVENT
6. ALTER ANY EVENT SESSION ENABLE
7. ALTER ANY EVENT SESSION DISABLE
8. ALTER ANY EVENT SESSION ADD TARGET
9. ALTER ANY EVENT SESSION DROP TARGET

All these permissions are under the same parent-permission: **ALTER** **ANY** EVENT SESSION

##### Database level:

1. CREATE ANY DATABASE EVENT SESSION
2. DROP ANY DATABASE EVENT SESSION
3. ALTER ANY DATABASE EVENT SESSION OPTION
4. ALTER ANY DATABASE EVENT SESSION ADD EVENT
5. ALTER ANY DATABASE EVENT SESSION DROP EVENT
6. ALTER ANY DATABASE EVENT SESSION ENABLE
7. ALTER ANY DATABASE EVENT SESSION DISABLE
8. ALTER ANY DATABASE EVENT SESSION ADD TARGET
9. ALTER ANY DATABASE EVENT SESSION DROP TARGET

All these permissions are under the same parent-permission: **ALTER** **ANY** DATABASE EVENT SESSION

 Scenarios that the new permissions enable.

**Preparing XEvent sessions**  
It is now possible to prepare XEvent sessions and only grant other Users to start & stop those prepared sessions.

**Allow users to solely add events or targets**  
By granting ALTER ANY EVENT SESSION ADD EVENT/ADD TARGET, Users can be allowed to add additional events to capture and add new targets – while at the same time be prevented to remove existing targets (together with the captured data therein) or events.

**Allow changing configuration but not what is captured**  
By granting ALTER ANY EVENT SESSION OPTION, users can change memory usage, latency, and similar options, but not influence what is captured and where. – For a list of session options look here: [ALTER EVENT SESSION (Transact-SQL) - SQL Server | Microsoft Docs](https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-event-session-transact-sql?view=sql-server-ver16#syntax) for “*event\_session\_options*”.

**Adding and modifying predicates** falls under ALTER ANY EVENT SESSION **ADD EVENT**, since that is part of the event configuration. The same applies to **adding *Actions***.

To view data from XEvent sessions, depending on the target type used (memory or file), either DMVs using XQuery have to be queried while for the file target a system function is used ([Targets for Extended Events in SQL Server - SQL Server | Microsoft Docs](https://docs.microsoft.com/en-us/sql/relational-databases/extended-events/targets-for-extended-events-in-sql-server?view=sql-server-ver16)). All of those were covered by the VIEW SERVER STATE-permission and from now on under the VIEW SERVER PERFORMANCE STATE-permission respectively the VIEW DATABASE PERFORMANCE STATE-permission. This is part of the work on System Metadata permission from the former topic.

### Security-related objects

The last bucket of new permissions covers a few security-related objects.

Here are the 4 new permissions:

##### CREATE LOGIN

This permission is a subset of ALTER ANY LOGIN and allows for delegating the creation of new Logins while at the same time preventing the user with this permission could also change passwords of existing Logins and hence abuse his powers for Elevation of Privilege attacks.

##### VIEW ANY ERROR LOG

This quite simply allows the grantee to read the error logs of SQL Server (and SQL Server Agent). This is done via 3 procedures: *sys.xp\_enumerrorlogs, sys.sp\_readerrorlog and sys.xp\_readerrorlog*. Before SQL Server 2022, the VIEW SERVER STATE permission which also covers DMVs was the minimal required permission.

##### VIEW SERVER SECURITY AUDIT

This is very useful for Security Auditors who should not have the ability to also change existing SQL Auditing definitions. With this permission, the Audit Log can be read via the system function [sys.fn\_get\_audit\_file (Transact-SQL) - SQL Server | Microsoft Docs](https://docs.microsoft.com/en-us/sql/relational-databases/system-functions/sys-fn-get-audit-file-transact-sql?view=sql-server-ver16). Beforehand, the CONTROL permission on the server was required.

For the database level Auditing data, we added another separate permission:

##### VIEW DATABASE SECURITY AUDIT

### Backwards-compatibility

By adding the new permissions as alternative, lower permissions, we ensure backwards-compatibility: Existing scripts do not break because all existing Permission-assignments (in the above example: VIEW SERVER STATE) work as before. But those who have a need to assign more granular permissions now can differentiate between security-related system metadata and “everything else”.

### Additional Reading

* [New granular permissions for SQL Server 2022 and Azure SQL to improve adherence with PoLP - Microsoft Community Hub](https://techcommunity.microsoft.com/t5/sql-server-blog/new-granular-permissions-for-sql-server-2022-and-azure-sql-to/ba-p/3607507)

# Availability

## Contained availability group.

A contained availability group is an Always On availability group that supports:

* managing metadata objects (users, logins, permissions, SQL Agent jobs etc.) at the availability group level in addition to the instance level.
* specialized contained system databases within the availability group.

### Overview

Always On availability groups generally consist of one or more user databases intended to operate as a coordinated group, and which are replicated on some number of nodes in a cluster. When there is a failure in the node, or in the health of SQL Server on the node that hosts the primary copy, the group of databases are moved as a unit to another replica node in the availability group. All the user databases are kept in sync across all replicas of the availability group, either in synchronous or asynchronous mode.

This works well for applications that only interact with that set of user databases, but there are challenges when applications also rely on objects such as users, logins, permissions, agent jobs, etc., which are stored in one of the system databases (master or msdb). For the applications to function smoothly and predictably, the admin must *manually* ensure that any change to these objects is duplicated across all replicas in the availability group. If a new instance is brought into the availability group, the databases can be automatically or manually seeded in a straightforward process, but then all the system database customizations must be reconfigured on the new instance to match the other replicas.

Contained availability groups extend the concept of the group of databases being replicated to include relevant portions of the master and msdb databases. Think of it as the execution context for applications using the contained availability group. The idea is that the contained AG environment includes settings that would affect the application relying on them. As such, the contained AG environment concerns all databases the application interacts with, the authentication it uses (logins, users, permissions), any scheduled jobs that it expects to be running, and other configuration settings that impact the application.

This is different from contained databases, which use a different mechanism for the user accounts, storing the user information within the database itself. Contained databases only replicate logins and users, and the scope of the replicated login or user is limited to that single database (and its replicas).

In contrast, in a contained availability group, you can create users, logins, permissions, etc. at the availability group level, and they will *automatically* be consistent across replicas in the availability group, as well as consistent across databases within that contained availability group. This saves the admin from having to manually make these changes themselves.

### Differences

There are some practical differences to consider when working with contained availability groups, such as the creation of contained system databases, and forcing the connection at the contained availability group level, rather than connecting at the instance level.

### Contained System Databases

Each contained availability group has its own master and msdb system databases, named after the name of the availability group. For example, in contained availability group MyContainedAG, you will have databases named MyContainedAG\_master and MyContainedAG\_msdb. These system databases are automatically seeded into new replicas and updates are replicated to these databases just like any other database in an availability group. This means that when you add an object such as a login, or agent job while connected to the contained availability group, when the contained availability group fails over to another instance, connecting to the contained availability group, you will still see the agent jobs, and be able to authenticate using the login created in the contained availability group.

The system databases in a newly created contained availability group are not copies from the instance where the CREATE AVAILABILITY GROUP command is run. They are initially empty templates without any data. Immediately after creation, the admin accounts on the instance creating the contained AG are copied into Contained Master. That way the admin can log into the contained AG and set up the rest of the configuration. If you've created local users or configurations in your instance, they will not automatically appear when you create your contained system databases, and they will not be visible when you connect to the contained availability group. You need to manually re-create them in the contained system databases within the context of the contained availability group. The exception to this is that all the logins in the sysadmin role in the parent instance are copied into the new AG specific master DB.

### Connect (Contained environment)

It's important to distinguish the difference between connecting to the instance and connecting to the contained availability group. The only way to access the environment of the contained availability group is to connect to the contained availability group listener, or to connect to a database which is in the contained availability group. i.e.

"Persist Security Info=False;

User ID=MyUser;Password=\*\*\*\*\*;

Initial Catalog=MyContainedDatabase;

Server=MyServer;"

Where MyContainedDatabase is a database within the contained availability group which you wish to interact with.

**This means that you must create a listener for the contained availability group to effectively use a contained availability group.** If you connect to one of the *instances* hosting the contained availability group rather than *directly to the contained availability group through the listener*, you will be in the environment of the instance, and not the contained availability group.

For example, if your availability group MyContainedAG is hosted on server SERVER\MSSQLSERVER, and instead of connecting to the listener MyContainedAG\_Listener, you connect to the instance using SERVER\MSSQLSERVER, you will be in the environment of the instance, and not in the environment of MyContainedAG. This means you will be subject to the contents (users, permissions, jobs, etc.) that are found in the system databases of the instance. To access the contents found in the contained system databases of the contained availability group, connect to the contained availability group listener (MyContainedAG\_Listener, for example) instead. When you are connected to the instance through the contained availability group listener, when you interact with master, you are redirected to the contained master database (for example, MyContainedAG\_master`).

### Read-only routing and contained availability groups.

If you have configured read-only routing to redirect connections with read intent to a secondary replica (see [Configure read-only routing for an Always On availability group](https://learn.microsoft.com/en-us/sql/database-engine/availability-groups/windows/configure-read-only-routing-for-an-availability-group-sql-server?view=sql-server-ver16)) and you wish to connect using a login which is created in the contained availability group only, there are some additional considerations:

* You must specify a database which is part of the contained availability group in the connection string.
* The user specified in the connection string must have permission to access the database(s) in the contained availability group.

For example, in the following connection string, where AdventureWorks is a database within the contained availability group which has MyContainedListener, and where *MyUser* is a user defined in the contained availability group and none of the participating instances:

"Persist Security Info=False;

User ID=MyUser;Password=\*\*\*\*\*;

Initial Catalog=AdventureWorks;

Server=MyContainedListener;

ApplicationIntent=ReadOnly"

This connection string would get you connected to the readable secondary which is part of the ReadOnly Routing configuration, and you would be within the context of the contained availability group.

### Differences between connecting to the instance and connecting to the contained availability group.

* When connected to contained AG, users will only see databases in the contained AG, plus tempdb.
* At instance level, contained AG master and msdb names will be [contained AG]\_master, and [contained AG]\_msdb. Inside contained AG, their names are master and msdb.
* Database ID for contained AG master is 1 from inside contained AG, but something else when connected to the instance.
* While users will not see databases outside of the contained AG in sys. databases when connected in a contained AG connection, they will be able to access those databases by three-part name or through the *use* command.
* Server configuration through sp\_configure can be read from contained AG connection but can only be written from instance level.
* From contained AG connections, sysadmin can perform instance level operations, such as shutting down SQL Server.
* Most DB level, end point level, or AG level operations can only be performed from instance connections, not contained AG connections.

### Additional Reading

* [What is a contained availability group? - SQL Server Always On | Microsoft Learn](https://learn.microsoft.com/en-us/sql/database-engine/availability-groups/windows/contained-availability-groups-overview?view=sql-server-ver16)

## Distributed availability group

A distributed availability group (AG) is a special type of availability group that spans two separate availability groups. Distributed availability groups are available starting with SQL Server 2016. Starting with sql server 2022, Now using multiple TCP connections for better network bandwidth utilization across a remote link with long tcp latencies.

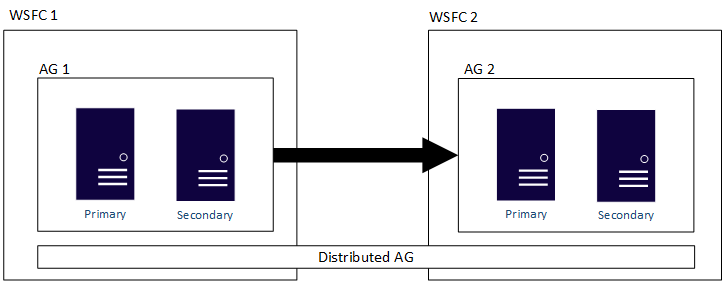
### Overview

A distributed availability group is a special type of availability group that spans two separate availability groups. The availability groups that participate in a distributed availability group don't need to be in the same location. They can be physical, virtual, on-premises, in the public cloud, or anywhere that supports an availability group deployment. This includes cross-domain and even cross-platform - such as between an availability group hosted on Linux and one hosted on Windows. As long as two availability groups can communicate, you can configure a distributed availability group with them.

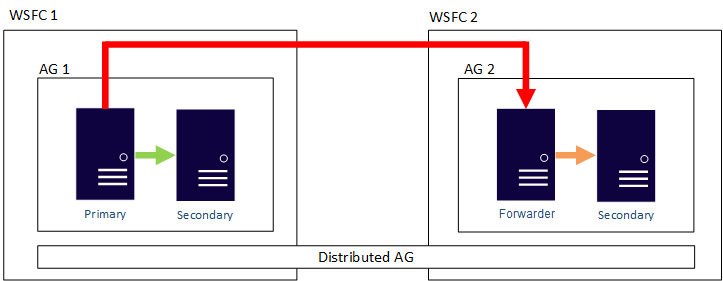
A traditional availability group has resources configured in a Windows Server Failover Cluster (WSFC) or if on Linux, Pacemaker. A distributed availability group doesn't configure anything in the underlying cluster (WSFC or Pacemaker). Everything about it is maintained within SQL Server. To learn how to view information for a distributed availability group, see [Viewing distributed availability group information](https://learn.microsoft.com/en-us/sql/database-engine/availability-groups/windows/distributed-availability-groups?view=sql-server-ver16#monitor-health).

A distributed availability group requires that the underlying availability groups have a listener. Rather than provide the underlying server name for a standalone instance (or in the case of a SQL Server failover cluster instance [FCI], the value associated with the network name resource) as you would with a traditional availability group, you specify the configured listener for the distributed availability group with the parameter ENDPOINT\_URL when you create it. Although each underlying availability group of the distributed availability group has a listener, a distributed availability group has no listener.

The following figure shows a high-level view of a distributed availability group that spans two availability groups (AG 1 and AG 2), each configured on its own WSFC. The distributed availability group has a total of four replicas, with two in each availability group. Each availability group can support up to the maximum number of replicas, so a distributed availability group can have up to 18 total replicas.



You can configure the data movement in distributed availability groups as synchronous or asynchronous. However, data movement is slightly different within distributed availability groups compared to a traditional availability group. Although each availability group has a primary replica, there is only one copy of the databases participating in a distributed availability group that can accept inserts, updates, and deletions. As shown in the following figure, AG 1 is the primary availability group. Its primary replica sends transactions to both the secondary replicas of AG 1 and the primary replica of AG 2. The primary replica of AG 2 is also known as a *forwarder*. A forwarder is a primary replica in a secondary availability group in a distributed availability group. The forwarder receives transactions from the primary replica in the primary availability group and forwards them to the secondary replicas in its own availability group. The forwarder then keeps the secondary replicas of AG 2 updated.



The only way to make AG 2's primary replica accept inserts, updates, and deletions, is to manually fail over the distributed availability group from AG 1. In the preceding figure, because AG 1 contains the writeable copy of the database, issuing a failover makes AG 2 the availability group that can handle inserts, updates, and deletions. For information about how to fail over one distributed availability group to another.

PolyBase enables your SQL Server instance to query data with T-SQL directly from SQL Server, Oracle, Teradata, MongoDB, Hadoop clusters, Cosmos DB without separately installing client connection software. You can also use the generic ODBC connector to connect to additional providers using third-party ODBC drivers. PolyBase allows T-SQL queries to join the data from external sources to relational tables in an instance of SQL Server.

A key use case for data virtualization with the PolyBase feature is to allow the data to stay in its original location and format. You can virtualize the external data through the SQL Server instance, so that it can be queried in place like any other table in SQL Server. This process minimizes the need for ETL processes for data movement. This data virtualization scenario is possible with the use of PolyBase connectors.

Polybase came in SQL 2016, but we now have expanded the number of data sources.

For NoSQL, Relational, and Hadoop, all client software is built in. ODBC is “bring your own driver” but opens the possibilities.

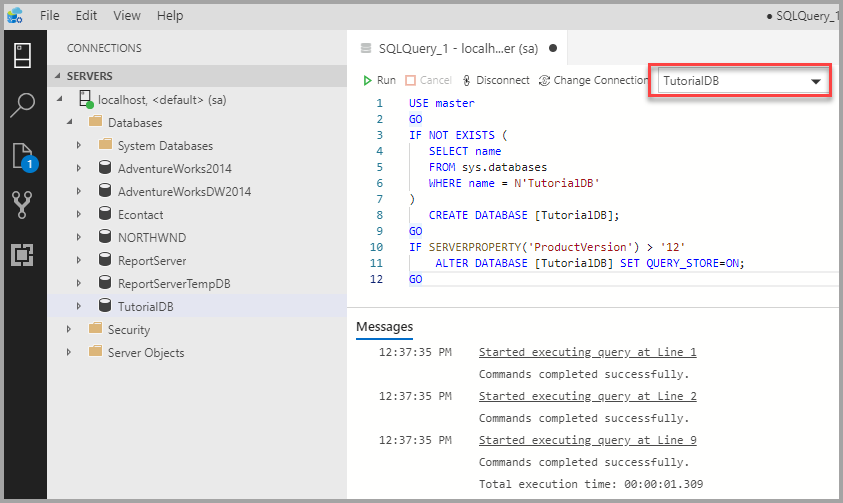
Note as of CTP 2.5, Polybase for Linux only supports NoSQL and Relational databases.

Some functionality of the PolyBase feature is in private preview for Azure SQL managed instances, including the ability to query external data (Parquet files) in Azure Data Lake Storage (ADLS) Gen2. Private preview includes access to client libraries and documentation for testing purposes that are not yet available publicly. If you are interested and ready to invest some time in trying out the functionalities and sharing your feedback and questions, please review the Azure SQL Managed Instance PolyBase Private Preview Guide.

# Tools

## Azure Data Studio

Azure Data Studio is a cross-platform database tool for data professionals using on-premises and cloud data platforms on Windows, macOS, and Linux. You can now  [Download and install Azure Data Studio](https://learn.microsoft.com/en-us/sql/azure-data-studio/download-azure-data-studio?view=sql-server-ver16) for sql server 2022. Azure Data Studio offers a modern editor experience with IntelliSense, code snippets, source control integration, and an integrated terminal. It's engineered with the data platform user in mind, with built-in charting of query result sets and customizable dashboards.



The source code for Azure Data Studio and its data providers is available on GitHub under a source code EULA that provides rights to modify and use the software, but not to redistribute it or host it in a cloud service. For more information, see [Azure Data Studio FAQ](https://docs.microsoft.com/en-us/sql/azure-data-studio/faq?view=sql-server-ver15).

* [Download and Install Azure Data Studio](https://docs.microsoft.com/en-us/sql/azure-data-studio/download-azure-data-studio?view=sql-server-ver15)

### Feature comparison with SQL Server Management Studio (SSMS)

**Use Azure Data Studio if you:**

* Are mostly editing or executing queries.
* Need the ability to quickly chart and visualize result sets.
* Can execute most administrative tasks via the integrated terminal using sqlcmd or PowerShell.
* Have minimal need for wizard experiences.
* Do not need to do deep administration or platform related configuration.
* Need to run on macOS or Linux.

**Use SQL Server Management Studio if you:**

* Are doing complex administration or platform configuration.
* Are security management, including user management, vulnerability assessment, and configuration of security features.
* I need to make use of performance tuning advisors and dashboards.
* Use database diagrams and table designers.
* Need access to Registered Servers.
* Make use of live query stats or client statistics.

### Extensions

Extensions in Azure Data Studio provide an easy way to add more functionality to the base Azure Data Studio installation.

Extensions are provided by the Azure Data Studio team (Microsoft), as well as the third-party community (you!).

### Notebooks

Jupyter Notebook is an open-source web application that allows you to create and share documents containing live code, equations, visualizations, and narrative text. Usage includes data cleaning and transformation, numerical simulation, statistical modeling, data visualization, and machine learning.

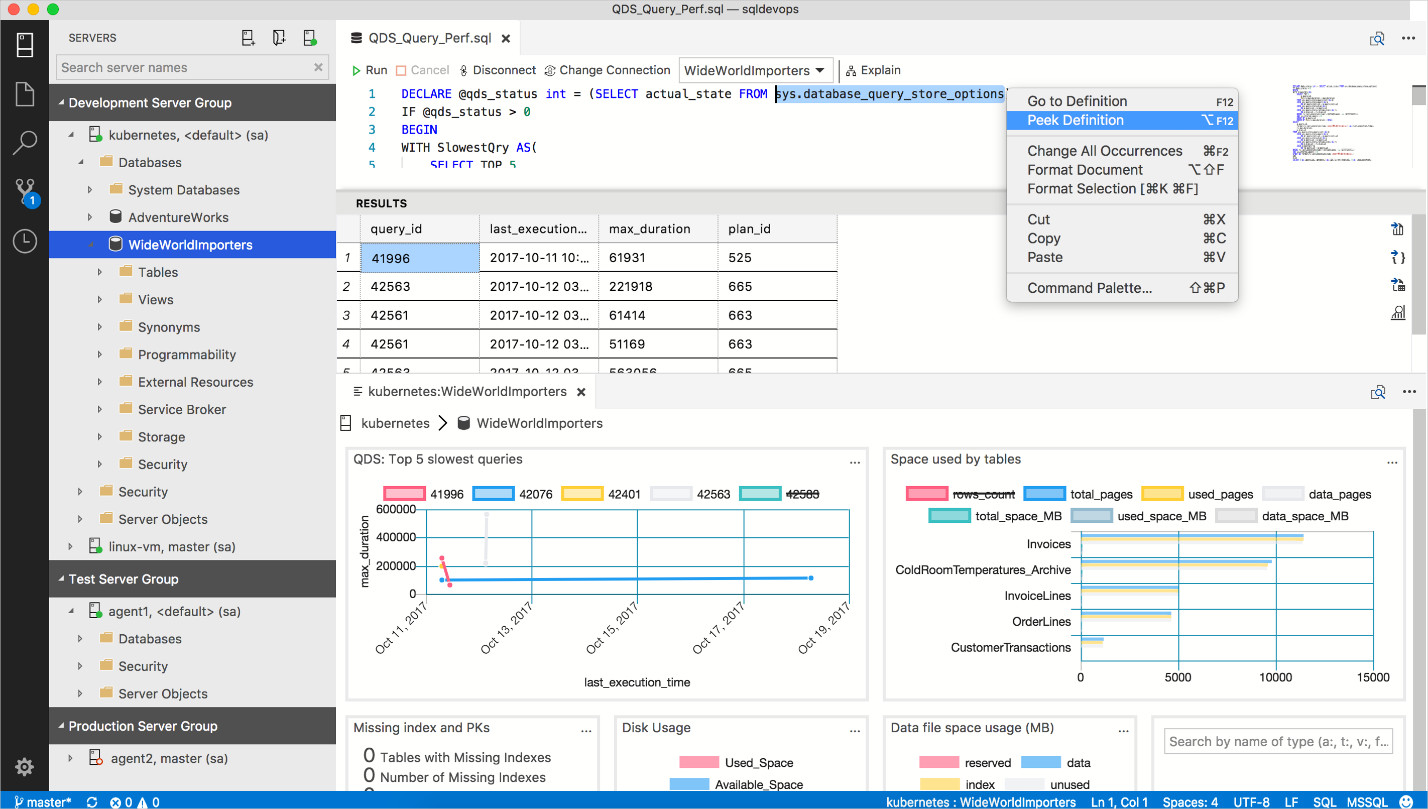
### Multiple Kernels

Azure Data Studio notebooks support several different kernels, including SQL Server, Python, PySpark, R, and others. Each kernel supports a different language in the code cells of your notebook. For example, when connected to the SQL Server kernel, you can enter and run T-SQL statements in a notebook code cell.

## Dashboards & Insights Widgets

Insight widgets take the Transact-SQL (T-SQL) queries you use to monitor servers & databases and turn them into insightful visualizations.

Insights are customizable charts and graphs that you add to server and database monitoring dashboards. View at-a-glance insights of your servers and databases, then drill into more details, and launch management actions that you define.



For more information, see [What is Azure Data Studio - Azure Data Studio | Microsoft Docs](https://docs.microsoft.com/en-us/sql/azure-data-studio/what-is-azure-data-studio?view=sql-server-ver15).

## SQL Server Management Studio

SQL Server Management Studio (SSMS) is an integrated environment for managing any SQL infrastructure, from SQL Server to Azure SQL Database. SSMS provides tools to configure, monitor, and administer instances of SQL Server and databases. Use SSMS to deploy, monitor, and upgrade the data-tier components used by your applications, and build queries and scripts.

Use SSMS to query, design, and manage your databases and data warehouses, wherever they are - on your local computer, or in the cloud.

The SSMS version 19.0 is now available and is the recommended version of SSMS for SQL Server 2022 (16.x). installation doesn't upgrade or replace SSMS versions 17.x or earlier. SSMS 18.x installs side by side with previous versions, so both versions are available for use. However, if you have a preview version of SSMS 18.x installed, you must uninstall it before installing SSMS 19.0. You can see if you have the preview version by going to the Help > About window.

Note

Beginning with SQL Server Management Studio (SSMS) 18.7, Azure Data Studio is automatically installed alongside SSMS.

For more information, see [SQL Server Management Studio (SSMS) - SQL Server Management Studio (SSMS) | Microsoft Docs](https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15).

## SSMS Life Cycle

Beginning with version 19.0, all security updates, critical updates, hotfixes, as well as any new features will be released only in the latest point version of the major version. Once a new version of SSMS is released for public, whether it's a point version within a major version or a major version itself, all prior versions are out of support, as per the [Modern Lifecycle Policy](https://support.microsoft.com/help/30881/modern-lifecycle-policy).

In December 2021, releases of SSMS prior to 18.6 will no longer authenticate to Database Engines through Azure Active Directory with MFA. To continue utilizing Azure Active Directory authentication with MFA, you need [**SSMS 18.6 or later**](https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16).

Connectivity to Azure Analysis Services through Azure Active Directory with MFA requires [**SSMS 18.5.1 or later**](https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16).

For best experience, it is recommended for customers to install the most recent version via <https://aka.ms/ssms>.

For more information, see [SQL Server Management Studio (SSMS) Support Policy - SQL Server Management Studio (SSMS) | Microsoft Docs](https://docs.microsoft.com/en-us/sql/ssms/support-policy?view=sql-server-ver15).