Contents

Log File Viewer
  Open
  View Offline
  F1 Help
Transaction log
  Manage size
  Troubleshoot a full log (SQL Server Error 9002)
  Control Transaction Durability
Database Checkpoints
  Change the Target Recovery Time of a Database

Log File Viewer

5/3/2018 • 2 minutes to read • Edit Online

THIS TOPIC APPLIES TO: ✓ SQL Server ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Log File Viewer in SQL Server Management Studio is used to access information about errors and events that are captured in log files.

Benefits of using Log File Viewer

You can view SQL Server log files from a local or remote instance of SQL Server when the target instance is offline or cannot start. You can access the offline log files from Registered Servers, or programmatically through WMI and WQL (WMI Query Language) queries. For more information, see View Offline Log Files. Following are the types of log files you can access using Log File Viewer:

- Audit Collection
- Data Collection
- Database Mail
- Job History
- Maintenance Plans
- Remote Maintenance Plans
- SQL Server
- SQL Server Agent
- Windows NT (These are Windows events that can also be accessed from Event Viewer.)

Log File Viewer Tasks

TASK DESCRIPTION	TOPIC
Describes how to open Log File Viewer depending on the information that you want to view.	Open Log File Viewer
Describes how to view offline log files through registered servers and how to verify WMI permissions.	View Offline Log Files
Provides Log File Viewer F1 Help.	Log File Viewer F1 Help

See Also

SQL Server Audit (Database Engine)
SQL Server Agent Error Log

Open Log File Viewer

5/3/2018 • 2 minutes to read • Edit Online

THIS TOPIC APPLIES TO: ✓ SQL Server ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

You can use Log File Viewer in SQL Server Management Studio to access information about errors and events that are captured in the following logs:

- Audit Collection
- Data Collection
- Database Mail
- Job History
- SQL Server
- SQL Server Agent
- Windows events (These Windows events can also be accessed from Event Viewer.)

  Beginning in SQL Server 2012 (11.x), you can use Registered Servers to view SQL Server log files from local or remote instances of SQL Server. By using Registered Servers, you can view the log files when the instances are either online or offline. For more information about online access, see the procedure "To view online log files from Registered Servers" later in this topic. For more information about how to access offline SQL Server log files, see View Offline Log Files.

  You can open Log File Viewer in several ways, depending on the information that you want to view.

Permissions

To access log files for instances of SQL Server that are online, this requires membership in the securityadmin fixed server role.

To access log files for instances of SQL Server that are offline, you must have read access to both the **Root\Microsoft\SqlServer\ComputerManagement10** WMI namespace, and to the folder where the log files are stored. For more information, see the Security section of the topic View Offline Log Files.

Security

Requires membership in the securityadmin fixed server role.

View Log Files

To view logs that are related to general SQL Server activity

1. In Object Explorer, expand **Management**.
2. Do either of the following:
   - Right-click **SQL Server Logs**, point to **View**, and then click either **SQL Server Log** or **SQL Server and Windows Log**.
   - Expand **SQL Server Logs**, right-click any log file, and then click **View SQL Server Log**. You can also double-click any log file.

     The logs include **Database Mail**, **SQL Server**, **SQL Server Agent**, and **Windows NT**.

To view logs that are related to jobs

- In Object Explorer, expand **SQL Server Agent**, right-click **Jobs**, and then click **View History**.

  The logs include **Database Mail**, **Job History**, and **SQL Server Agent**.

To view logs that are related to maintenance plans

- In Object Explorer, expand **Management**, right-click **Maintenance Plans**, and then click **View History**.

  The logs include **Database Mail**, **Job History**, **Maintenance Plans**, **Remote Maintenance Plans**, and **SQL Server Agent**.

To view logs that are related to Data Collection

- In Object Explorer, expand **Management**, right-click **Data Collection**, and then click **View Logs**.

  The logs include **Data Collection**, **Job History**, and **SQL Server Agent**.

To view logs that are related to Database Mail

- In Object Explorer, expand **Management**, right-click **Database Mail**, and then click **View Database Mail Log**.

  The logs include **Database Mail, Job History**, **Maintenance Plans**, **Remote Maintenance Plans**, **SQL Server**, **SQL Server Agent**, and **Windows NT**.

To view logs that are related to audits collections

- In Object Explorer, expand **Security**, expand **Audits**, right-click an audit, and then click **View Audit Logs**.

  The logs include **Audit Collection** and **Windows NT**.

To view logs that are related to audits collections

- In Object Explorer, expand **Security**, expand **Audits**, right-click an audit, and then click **View Audit Logs**.

  The logs include **Audit Collection** and **Windows NT**.

See Also

Log File Viewer
SQL Server Audit (Database Engine)
View Offline Log Files

View Offline Log Files

5/3/2018 • 4 minutes to read • Edit Online

THIS TOPIC APPLIES TO: ✓ SQL Server ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Beginning in SQL Server 2012 (11.x), you can view SQL Server log files from a local or remote instance of SQL Server when the target instance is offline or cannot start.

You can access the offline log files from Registered Servers, or programmatically through WMI and WQL (WMI Query Language) queries.

NOTE

You can also use these methods to connect to an instance that is online, but for some reason, you cannot connect through a SQL Server connection.

Before you Begin

To connect to offline log files, an instance of SQL Server must be installed on the computer that you are using to view the offline log files, and on the computer where the log files that you want to view are located. If an instance of SQL Server is installed on both computers, you can view offline files for instances of SQL Server, and for instances that are running earlier versions of SQL Server on either computer.

If you are using Registered Servers, the instance that you want to connect to must be registered under **Local Server Groups** or under **Central Management Servers**. (The instance can be registered on its own or be a member of a server group.) For more information about how to add an instance of SQL Server to Registered Servers, see the following topics:

- Create or Edit a Server Group (SQL Server Management Studio)
- Register a Connected Server (SQL Server Management Studio)
- Create a Central Management Server and Server Group (SQL Server Management Studio)

  For more information about how to view offline log files programmatically through WMI and WQL queries, see the following topics:

- SqlErrorLogEvent Class (This topic shows how to retrieve values for logged events in a specified log file.)
- SqlErrorLogFile Class (This topic shows how to retrieve information about all SQL Server log files on a specified instance of SQL Server.)

Permissions

To connect to an offline log file, you must have the following permissions on both the local and remote computers:

- Read access to the **Root\Microsoft\SqlServer\ComputerManagement12** WMI namespace. By default, everyone has read access through the Enable Account permission. For more information, see the "To verify WMI permissions" procedure later in this section.
- Read permission to the folder that contains the error log files. By default the error log files are located in the following path (where <*Drive*> represents the drive where you installed SQL Server and <*InstanceName*> is the name of the instance of SQL Server):

<Drive>:\Program Files\Microsoft SQL Server\MSSQL13.<InstanceName>\MSSQL\Log

To verify WMI namespace security settings, you can use the WMI Control snap-in.

To verify WMI permissions

1. Open the WMI Control snap-in. To do this, do either of the following, depending on the operating system:
   - Click **Start**, type **wmimgmt.msc** in the **Start Search** box, and then press ENTER.
   - Click **Start**, click **Run**, type **wmimgmt.msc**, and then press ENTER.
2. By default, the WMI Control snap-in manages the local computer.

   If you want to connect to a remote computer, follow these steps:

   a. Right-click **WMI Control (Local)**, and then click **Connect to another computer**.

   b. In the **Change managed computer** dialog box, click **Another computer**.

   c. Enter the remote computer name, and then click **OK**.
3. Right-click **WMI Control (Local)** or **WMI Control (***RemoteComputerName***)**, and then click **Properties**.
4. In the **WMI Control Properties** dialog box, click the **Security** tab.
5. In the namespace tree, locate and then click the following namespace:

   Root\Microsoft\SqlServer\ComputerManagement10
6. Click **Security**.
7. Make sure that the account that will be used has the **Enable Account** permission. This permission allows Read access to WMI objects.

View Log Files

The following procedure shows how to view offline log files through Registered Servers. The procedure assumes the following:

The instance of SQL Server that you want to connect to is already registered in Registered Servers.

To view log files for instances that are offline

1. If you want to view offline log files on a local instance, make sure that you start SQL Server Management Studio with elevated permissions. To do this, when you start Management Studio, right-click **SQL Server Management Studio**, and then click **Run as administrator**.
2. In SQL Server Management Studio, on the **View** menu, click **Registered Servers**.
3. In the console tree, locate the instance on which you want to view the offline files.
4. Do one of the following:
   - If the instance is under **Local Server Groups**, expand **Local Server Groups**, expand the server group (if the instance is a member of a group), right-click the instance, and then click **View SQL Server Log**.
   - If the instance is the Central Management Server itself, expand **Central Management Servers**, right-click the instance, point to **Central Management Server Actions**, and then click **View SQL Server Log**.
   - If the instance is under **Central Management Servers**, expand **Central Management Servers**, expand the Central Management Server, right-click the instance (or expand a server group and right-click the instance), and then click **View SQL Server Log**.

5. If you are connecting to a local instance, the connection is made using the current user credentials.

   If you are connecting to a remote instance, in the **Log File Viewer - Connect As** dialog box, do either of the following:

   - To connect as the current user, make sure that the **Connect as another user** check box is cleared, and then click **OK**.

   - To connect as another user, select the **Connect as another user** check box, and then click **Set User**. When you are prompted, enter the user credentials (with the user name in the format *domain_name*\user_name), click **OK**, and then click **OK** again to connect.

   NOTE

   If the log files take too long to load, you can click **Stop** on the Log File Viewer toolbar.

See Also

Log File Viewer

Log File Viewer F1 Help

5/3/2018 • 2 minutes to read • Edit Online

THIS TOPIC APPLIES TO: ✓ SQL Server ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Log File Viewer displays log information from many different components. When Log File Viewer is open, use the **Select logs** pane to select the logs you want to display. Each log displays columns appropriate to that kind of log.

The logs that are available depend on how Log File Viewer is opened. For more information, see Open Log File Viewer.

The number of rows that are displayed for audit logs can be configured on the **SQL Server Object Explorer/Commands** page of the **Tools/Options** dialog box. For descriptions of the columns that are displayed for audit logs, see sys.fn_get_audit_file (Transact-SQL).

Options

Load Log
Open a dialog box where you can specify a log file to load.

Export
Open a dialog box that lets you export the information that is shown in the **Log file summary** grid to a text file.

Refresh
Refresh the view of the selected logs. The **Refresh** button rereads the selected logs from the target server while applying any filter settings.

Filter
Open a dialog box that lets you specify settings that are used to filter the log file, such as **Connection**, **Date**, or other **General** filter criteria.

Search
Search the log file for specific text. Searching with wildcard characters is not supported.

Stop
Stops loading the log file entries. For example, you can use this option if a remote or offline log file takes a long time to load, and you only want to view the most recent entries.

Log file summary
This information panel displays a summary of the log file filtering. If the file is not filtered, you will see the following text, **No filter applied**. If a filter is applied to the log, you will see the following text, **Filter log entries where:** <filter criteria>.

Selected row details
Select a row to display additional details about the selected event row at the bottom of the page. The columns can be reordered by dragging them to new locations in the grid. The columns can be resized by dragging the column separator bars in the grid header to the left or right. Double-click the column separator bars in the grid header to automatically size the column to the content width.

Instance
The name of the instance on which the event occurred. This is displayed as *computer name\instance name*.

Frequently Displayed Columns

Date
Displays the date of the event.

Source
Displays the source feature from which the event is created, such as the name of the service (MSSQLSERVER, for example). This does not appear for all log types.

Message
Displays any messages associated with the event.

Log Type
Displays the type of log to which the event belongs. All selected logs appear in the log file summary window.

Log Source
Displays a description of the source log in which the event is captured.

Permissions

To access log files for instances of SQL Server that are online, this requires membership in the securityadmin fixed server role.

To access log files for instances of SQL Server that are offline, you must have read access to both the **Root\Microsoft\SqlServer\ComputerManagement10** WMI namespace, and to the folder where the log files are stored. For more information, see the Security section of the topic View Offline Log Files.

See Also

Log File Viewer
Open Log File Viewer
View Offline Log Files

The Transaction Log (SQL Server)

5/3/2018 • 11 minutes to read • Edit Online

THIS TOPIC APPLIES TO: ✓ SQL Server ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Every SQL Server database has a transaction log that records all transactions and the database modifications made by each transaction.

The transaction log is a critical component of the database. If there is a system failure, you will need that log to bring your database back to a consistent state.

For information about the transaction log architecture and internals, see the SQL Server Transaction Log Architecture and Management Guide.

WARNING

Never delete or move this log unless you fully understand the ramifications of doing so.

TIP

Known good points from which to begin applying transaction logs during database recovery are created by checkpoints. For more information, see Database Checkpoints (SQL Server).

Operations supported by the transaction log

The transaction log supports the following operations:

- Individual transaction recovery.
- Recovery of all incomplete transactions when SQL Server is started.
- Rolling a restored database, file, filegroup, or page forward to the point of failure.
- Supporting transactional replication.
- Supporting high availability and disaster recovery solutions: Always On availability groups, database mirroring, and log shipping.

Individual transaction recovery

If an application issues a `ROLLBACK` statement, or if the Database Engine detects an error such as the loss of communication with a client, the log records are used to roll back the modifications made by an incomplete transaction.

Recovery of all incomplete transactions when SQL Server is started

If a server fails, the databases may be left in a state where some modifications were never written from the buffer cache to the data files, and there may be some modifications from incomplete transactions in the data files. When an instance of SQL Server is started, it runs a recovery of each database. Every modification recorded in the log which may not have been written to the data files is rolled forward. Every incomplete transaction found in the transaction log is then rolled back to make sure the integrity of the database is preserved.

Rolling a restored database, file, filegroup, or page forward to the point of failure

After a hardware loss or disk failure affecting the database files, you can restore the database to the point of failure. You first restore the last full database backup and the last differential database backup, and then restore the subsequent sequence of the transaction log backups to the point of failure.

As you restore each log backup, the Database Engine reapplies all the modifications recorded in the log to roll forward all the transactions. When the last log backup is restored, the Database Engine then uses the log information to roll back all transactions that were not complete at that point.

Supporting transactional replication

The Log Reader Agent monitors the transaction log of each database configured for transactional replication and copies the transactions marked for replication from the transaction log into the distribution database. For more information, see How Transactional Replication Works.

Supporting high availability and disaster recovery solutions

The standby-server solutions, Always On availability groups, database mirroring, and log shipping, rely heavily on the transaction log.

In an **Always On availability groups scenario**, every update to a database, the primary replica, is immediately reproduced in separate, full copies of the database, the secondary replicas. The primary replica sends each log record immediately to the secondary replicas which applies the incoming log records to availability group databases, continually rolling it forward. For more information, see Always On Failover Cluster Instances

In a **log shipping scenario**, the primary server sends the active transaction log of the primary database to one or more destinations. Each secondary server restores the log to its local secondary database. For more information, see About Log Shipping.

In a **database mirroring scenario**, every update to a database, the principal database, is immediately reproduced in a separate, full copy of the database, the mirror database. The principal server instance sends each log record immediately to the mirror server instance which applies the incoming log records to the mirror database, continually rolling it forward. For more information, see Database Mirroring.

Transaction Log characteristics

Characteristics of the SQL Server Database Engine transaction log:

- The transaction log is implemented as a separate file or set of files in the database. The log cache is managed separately from the buffer cache for data pages, which results in simple, fast, and robust code within the SQL Server Database Engine. For more information, see Transaction Log Physical Architecture.
- The format of log records and pages is not constrained to follow the format of data pages.
- The transaction log can be implemented in several files. The files can be defined to expand automatically by setting the `FILEGROWTH` value for the log. This reduces the potential of running out of space in the transaction log, while at the same time reducing administrative overhead. For more information, see ALTER DATABASE (Transact-SQL) File and Filegroup Options.
- The mechanism to reuse the space within the log files is quick and has minimal effect on transaction throughput.

For information about the transaction log architecture and internals, see the SQL Server Transaction Log Architecture and Management Guide.

Transaction log truncation

Log truncation frees space in the log file for reuse by the transaction log. You must regularly truncate your transaction log to keep it from filling the alotted space. Several factors can delay log truncation, so monitoring log size matters. Some operations can be minimally logged to reduce their impact on transaction log size.

Log truncation deletes inactive virtual log files (VLFs) from the logical transaction log of a SQL Server database, freeing space in the logical log for reuse by the Physical transaction log. If a transaction log is never truncated, it will eventually fill all the disk space allocated to physical log files.

To avoid running out of space, unless log truncation is delayed for some reason, truncation occurs automatically after the following events:

- Under the simple recovery model, after a checkpoint.
- Under the full recovery model or bulk-logged recovery model, if a checkpoint has occurred since the previous backup, truncation occurs after a log backup (unless it is a copy-only log backup).

  For more information, see Factors that can delay log truncation, later in this topic.

NOTE

Log truncation does not reduce the size of the physical log file. To reduce the physical size of a physical log file, you must shrink the log file. For information about shrinking the size of the physical log file, see Manage the Size of the Transaction Log File.
However, keep in mind Factors that can delay log truncation. If the storage space is required again after a log shrink, the transaction log will grow again and by doing that, introduce performance overhead during log grow operations.

Factors that can delay log truncation

When log records remain active for a long time, transaction log truncation is delayed, and the transaction log can fill up, as we mentioned earlier in this long topic.

IMPORTANT

For information about how to respond to a full transaction log, see Troubleshoot a Full Transaction Log (SQL Server Error 9002).

Really, Log truncation can be delayed by a variety of reasons. Learn what, if anything, is preventing your log truncation by querying the **log_reuse_wait** and **log_reuse_wait_desc** columns of the sys.databases catalog view. The following table describes the values of these columns.

LOG_REUSE_WAIT VALUE	LOG_REUSE_WAIT_DESC VALUE	DESCRIPTION
0	NOTHING	Currently there are one or more reusable virtual log files (VLFs).
1	CHECKPOINT	No checkpoint has occurred since the last log truncation, or the head of the log has not yet moved beyond a virtual log file (VLF). (All recovery models) This is a routine reason for delaying log truncation. For more information, see Database Checkpoints (SQL Server).
2	LOG_BACKUP	A log backup is required before the transaction log can be truncated. (Full or bulk-logged recovery models only) When the next log backup is completed, some log space might become reusable.

LOG_REUSE_WAIT VALUE	LOG_REUSE_WAIT_DESC VALUE	DESCRIPTION
3	ACTIVE_BACKUP_OR_RESTORE	A data backup or a restore is in progress (all recovery models). If a data backup is preventing log truncation, canceling the backup operation might help the immediate problem.
4	ACTIVE_TRANSACTION	A transaction is active (all recovery models): A long-running transaction might exist at the start of the log backup. In this case, freeing the space might require another log backup. Note that long-running transactions prevent log truncation under all recovery models, including the simple recovery model, under which the transaction log is generally truncated on each automatic checkpoint. A transaction is deferred. A *deferred transaction* is effectively an active transaction whose rollback is blocked because of some unavailable resource. For information about the causes of deferred transactions and how to move them out of the deferred state, see Deferred Transactions (SQL Server). Long-running transactions might also fill up tempdb's transaction log. Tempdb is used implicitly by user transactions for internal objects such as work tables for sorting, work files for hashing, cursor work tables, and row versioning. Even if the user transaction includes only reading data (`SELECT` queries), internal objects may be created and used under user transactions. Then the tempdb transaction log can be filled.
5	DATABASE_MIRRORING	Database mirroring is paused, or under high-performance mode, the mirror database is significantly behind the principal database. (Full recovery model only) For more information, see Database Mirroring (SQL Server).
6	REPLICATION	During transactional replications, transactions relevant to the publications are still undelivered to the distribution database. (Full recovery model only) For information about transactional replication, see SQL Server Replication.

LOG_REUSE_WAIT VALUE	LOG_REUSE_WAIT_DESC VALUE	DESCRIPTION
7	DATABASE_SNAPSHOT_CREATION	A database snapshot is being created. (All recovery models) This is a routine, and typically brief, cause of delayed log truncation.
8	LOG_SCAN	A log scan is occurring. (All recovery models) This is a routine, and typically brief, cause of delayed log truncation.
9	AVAILABILITY_REPLICA	A secondary replica of an availability group is applying transaction log records of this database to a corresponding secondary database. (Full recovery model) For more information, see Overview of Always On Availability Groups (SQL Server).
10	—	For internal use only
11	—	For internal use only
12	—	For internal use only
13	OLDEST_PAGE	If a database is configured to use indirect checkpoints, the oldest page on the database might be older than the checkpoint log sequence number (LSN). In this case, the oldest page can delay log truncation. (All recovery models) For information about indirect checkpoints, see Database Checkpoints (SQL Server).
14	OTHER_TRANSIENT	This value is currently not used.

Operations that can be minimally logged

Minimal logging involves logging only the information that is required to recover the transaction without supporting point-in-time recovery. This topic identifies the operations that are minimally logged under the bulk-logged recovery model (as well as under the simple recovery model, except when a backup is running).

NOTE

Minimal logging is not supported for memory-optimized tables.

NOTE

Under the full recovery model, all bulk operations are fully logged. However, you can minimize logging for a set of bulk operations by switching the database to the bulk-logged recovery model temporarily for bulk operations. Minimal logging is more efficient than full logging, and it reduces the possibility of a large-scale bulk operation filling the available transaction log space during a bulk transaction. However, if the database is damaged or lost when minimal logging is in effect, you cannot recover the database to the point of failure.

The following operations, which are fully logged under the full recovery model, are minimally logged under the simple and bulk-logged recovery model:

- Bulk import operations (bcp, BULK INSERT, and INSERT... SELECT). For more information about when bulk import into a table is minimally logged, see Prerequisites for Minimal Logging in Bulk Import.

When transactional replication is enabled, `BULK INSERT` operations are fully logged even under the Bulk Logged recovery model.

- SELECT INTO operations.

When transactional replication is enabled, SELECT INTO operations are fully logged even under the Bulk Logged recovery model.

- Partial updates to large value data types, using the `.WRITE` clause in the UPDATE statement when inserting or appending new data. Note that minimal logging is not used when existing values are updated. For more information about large value data types, see Data Types (Transact-SQL).

- WRITETEXT and UPDATETEXT statements when inserting or appending new data into the **text**, **ntext**, and **image** data type columns. Note that minimal logging is not used when existing values are updated.

  WARNING

  The `WRITETEXT` and `UPDATETEXT` statements are **deprecated**; avoid using them in new applications.

- If the database is set to the simple or bulk-logged recovery model, some index DDL operations are minimally logged whether the operation is executed offline or online. The minimally logged index operations are as follows:

  - CREATE INDEX operations (including indexed views).

  - ALTER INDEX REBUILD or DBCC DBREINDEX operations.

    WARNING

    The `DBCC DBREINDEX` statement is **deprecated**; Do not use it in new applications.

  - DROP INDEX new heap rebuild (if applicable). Index page deallocation during a `DROP INDEX` operation is **always** fully logged.

Related tasks

Managing the transaction log

- Manage the Size of the Transaction Log File

- Troubleshoot a Full Transaction Log (SQL Server Error 9002)

Backing Up the Transaction Log (Full Recovery Model)

- Back Up a Transaction Log (SQL Server)

Restoring the Transaction Log (Full Recovery Model)

- Restore a Transaction Log Backup (SQL Server)

See also

SQL Server Transaction Log Architecture and Management Guide
Control Transaction Durability
Prerequisites for Minimal Logging in Bulk Import
Back Up and Restore of SQL Server Databases
Database Checkpoints (SQL Server)
View or Change the Properties of a Database
Recovery Models (SQL Server)
Transaction Log Backups (SQL Server)
sys.dm_db_log_info (Transact-SQL)
sys.dm_db_log_space_usage (Transact-SQL)

Manage the size of the transaction log file

5/3/2018 • 6 minutes to read • Edit Online

THIS TOPIC APPLIES TO: ✓ SQL Server ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This topic covers how to monitor SQL Server transaction log size, shrink the transaction log, add to or enlarge a transaction log file, optimize the **tempdb** transaction log growth rate, and control the growth of a transaction log file.

Monitor log space use

Monitor log space use by using sys.dm_db_log_space_usage. This DMV returns information about the amount of log space currently used, and indicates when the transaction log needs truncation.

For information about the current log file size, its maximum size, and the autogrow option for the file, you can also use the **size**, **max_size**, and **growth** columns for that log file in sys.database_files.

IMPORTANT

Avoid overloading the log disk. Ensure the log storage can withstand the IOPS and low latency requirements for your transactional load.

Shrink log file size

To reduce the physical size of a physical log file, you must shrink the log file. This is useful when you know that a transaction log file contains unused space. You can shrink a log file only while the database is online, and at least one virtual log file (VLF) is free. In some cases, shrinking the log may not be possible until after the next log truncation.

NOTE

Factors such as a long-running transaction, that keep VLFs active for an extended period, can restrict log shrinkage or even prevent the log from shrinking at all. For information, see Factors that can delay log truncation.

Shrinking a log file removes one or more VLFs that hold no part of the logical log (that is, *inactive VLFs*). When a transaction log file is shrunk, inactive VLFs are removed from the end of the log file to reduce the log to approximately the target size.

IMPORTANT

Before shrinking the transaction log, keep in mind Factors that can delay log truncation. If the storage space is required again after a log shrink, the transaction log will grow again and by doing that, introduce performance overhead during log growth operations. For more information, see the Recommendations in this topic.

Shrink a log file (without shrinking database files)

- DBCC SHRINKFILE (Transact-SQL)
- Shrink a File

Monitor log-file shrink events

- Log File Auto Shrink Event Class.

Monitor log space

- sys.dm_db_log_space_usage (Transact-SQL)
- sys.database_files (Transact-SQL) (See the **size**, **max_size**, and **growth** columns for the log file or files.)

Add or enlarge a log file

You can gain space by enlarging the existing log file (if disk space permits) or by adding a log file to the database, typically on a different disk. One transaction log file is sufficient unless log space is running out, and disk space is also running out on the volume that holds the log file.

- To add a log file to the database, use the `ADD LOG FILE` clause of the `ALTER DATABASE` statement. Adding a log file allows the log to grow.
- To enlarge the log file, use the `MODIFY FILE` clause of the `ALTER DATABASE` statement, specifying the `SIZE` and `MAXSIZE` syntax. For more information, see ALTER DATABASE (Transact-SQL) File and Filegroup options.

For more information, see the Recommendations in this topic.

Optimize tempdb transaction log size

Restarting a server instance resizes the transaction log of the **tempdb** database to its original, pre-autogrow size. This can reduce the performance of the **tempdb** transaction log.

You can avoid this overhead by increasing the size of the **tempdb** transaction log after starting or restarting the server instance. For more information, see tempdb Database.

Control transaction log file growth

Use the ALTER DATABASE (Transact-SQL) File and Filegroup options statement to manage the growth of a transaction log file. Note the following:

- To change the current file size in KB, MB, GB, and TB units, use the `SIZE` option.
- To change the growth increment, use the `FILEGROWTH` option. A value of 0 indicates that automatic growth is set to off and no additional space is permitted.
- To control the maximum the size of a log file in KB, MB, GB, and TB units or to set growth to UNLIMITED, use the `MAXSIZE` option.

For more information, see the Recommendations in this topic.

Recommendations

Following are some general recommendations when you are working with transaction log files:

- The automatic growth (autogrow) increment of the transaction log, as set by the `FILEGROWTH` option, must be large enough to stay ahead of the needs of the workload transactions. The file growth increment on a log file should be sufficiently large to avoid frequent expansion. A good pointer to properly size a transaction log is monitoring the amount of log occupied during:
  - The time required to execute a full backup, because log backups cannot occur until it finishes.
  - The time required for the largest index maintenance operations.
  - The time required to execute the largest batch in a database.

- When setting **autogrow** for data and log files using the `FILEGROWTH` option, it might be preferred to set it in **size** instead of **percentage**, to allow better control on the growth ratio, as percentage is an ever-growing amount.
  - Keep in mind that transaction logs cannot leverage Instant File Initialization, so extended log growth times are especially critical.
  - As a best practice, do not set the `FILEGROWTH` option value above 1,024 MB for transaction logs. The default values for `FILEGROWTH` option are:

VERSION	DEFAULT VALUES
Starting with SQL Server 2016 (13.x)	Data 64 MB. Log files 64 MB.
Starting with SQL Server 2005	Data 1 MB. Log files 10%.
Prior to SQL Server 2005	Data 10%. Log files 10%.

- A small growth increment can generate too many small VLFs and can reduce performance. To determine the optimal VLF distribution for the current transaction log size of all databases in a given instance, and the required growth increments to achieve the required size, see this script.
- A large growth increment can generate too few and large VLFs and can also affect performance. To determine the optimal VLF distribution for the current transaction log size of all databases in a given instance, and the required growth increments to achieve the required size, see this script.
- Even with autogrow enabled, you can receive a message that the transaction log is full, if it cannot grow fast enough to satisfy the needs of your query. For more information on changing the growth increment, see ALTER DATABASE (Transact-SQL) File and Filegroup options
- Having multiple log files in a database does not enhance performance in any way, because the transaction log files do not use proportional fill like data files in a same filegroup.
- Log files can be set to shrink automatically. However this is **not recommended**, and the **auto_shrink** database property is set to FALSE by default. If **auto_shrink** is set to TRUE, automatic shrinking reduces the size of a file only when more than 25 percent of its space is unused.
  - The file is shrunk either to the size at which only 25 percent of the file is unused space or to the original size of the file, whichever is larger.
  - For information about changing the setting of the **auto_shrink** property, see View or Change the Properties of a Database and ALTER DATABASE SET Options (Transact-SQL).

See also

BACKUP (Transact-SQL)
Troubleshoot a Full Transaction Log (SQL Server Error 9002)
Transaction Log Backups in the SQL Server Transaction Log Architecture and Management Guide
Transaction Log Backups (SQL Server)
ALTER DATABASE (Transact-SQL) File and Filegroup options

Troubleshoot a Full Transaction Log (SQL Server Error 9002)

5/3/2018 • 4 minutes to read • Edit Online

THIS TOPIC APPLIES TO: ✓ SQL Server ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This topic discusses possible responses to a full transaction log and suggests how to avoid it in the future.

When the transaction log becomes full, SQL Server Database Engine issues a **9002 error**. The log can fill when the database is online, or in recovery. If the log fills while the database is online, the database remains online but can only be read, not updated. If the log fills during recovery, the Database Engine marks the database as RESOURCE PENDING. In either case, user action is required to make log space available.

Responding to a full transaction log

The appropriate response to a full transaction log depends partly on what condition or conditions caused the log to fill.

To discover what is preventing log truncation in a given case, use the **log_reuse_wait** and **log_reuse_wait_desc** columns of the **sys.database** catalog view. For more information, see sys.databases (Transact-SQL). For descriptions of factors that can delay log truncation, see The Transaction Log (SQL Server).

> **IMPORTANT!!**
> If the database was in recovery when the 9002 error occurred, after resolving the problem, recover the database by using ALTER DATABASE *database_name* SET ONLINE.

Alternatives for responding to a full transaction log include:

- Backing up the log.
- Freeing disk space so that the log can automatically grow.
- Moving the log file to a disk drive with sufficient space.
- Increasing the size of a log file.
- Adding a log file on a different disk.
- Completing or killing a long-running transaction.

  These alternatives are discussed in the following sections. Choose a response that fits your situation best.

Back up the log

Under the full recovery model or bulk-logged recovery model, if the transaction log has not been backed up recently, backup might be what is preventing log truncation. If the log has never been backed up, you **must create two log backups** to permit the Database Engine to truncate the log to the point of the last backup. Truncating the log frees space for new log records. To keep the log from filling up again, take log backups frequently.

To create a transaction log backup

> **IMPORTANT**

> If the database is damaged, see Tail-Log Backups (SQL Server).

- Back Up a Transaction Log (SQL Server)
- SqlBackup (SMO)

Freeing disk space

You might be able to free disk space on the disk drive that contains the transaction log file for the database by deleting or moving other files. The freed disk space allows the recovery system to enlarge the log file automatically.

Move the log file to a different disk

If you cannot free enough disk space on the drive that currently contains the log file, consider moving the file to another drive with sufficient space.

> **IMPORTANT!!** Log files should never be placed on compressed file systems.

Move a log file

- Move Database Files

Increase log file size

If space is available on the log disk, you can increase the size of the log file. The maximum size for log files is two terabytes (TB) per log file.

Increase the file size

If autogrow is disabled, the database is online, and sufficient space is available on the disk, either:

- Manually increase the file size to produce a single growth increment.
- Turn on autogrow by using the ALTER DATABASE statement to set a non-zero growth increment for the FILEGROWTH option.

> **NOTE** In either case, if the current size limit has been reached, increase the MAXSIZE value.

Add a log file on a different disk

Add a new log file to the database on a different disk that has sufficient space by using ALTER DATABASE <database_name> ADD LOG FILE.

Add a log file

- Add Data or Log Files to a Database

Complete or kill a long-running transaction

Discovering long-running transactions

A very long-running transaction can cause the transaction log to fill. To look for long-running transactions, use one of the following:

- **sys.dm_tran_database_transactions.** This dynamic management view returns information about transactions at the database level. For a long-running transaction, columns of particular interest include the time of the first log record (database_transaction_begin_time), the current state of the transaction (database_transaction_state), and the log sequence number (LSN) of the begin record in the transaction log (database_transaction_begin_lsn).

- **DBCC OPENTRAN.** This statement lets you identify the user ID of the owner of the transaction, so you can potentially track down the source of the transaction for a more orderly termination (committing it rather than rolling it back).

Kill a transaction

Sometimes you just have to end the process; you may have to use the KILL statement. Please use this statement very carefully, especially when critical processes are running that you don't want to kill. For more information, see KILL (Transact-SQL)

See also

KB support article - A transaction log grows unexpectedly or becomes full in SQL Server ALTER DATABASE (Transact-SQL)
Manage the Size of the Transaction Log File
Transaction Log Backups (SQL Server)
sp_add_log_file_recover_suspect_db (Transact-SQL)

Control Transaction Durability

5/3/2018 • 9 minutes to read • Edit Online

THIS TOPIC APPLIES TO: ✓ SQL Server ✓ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

SQL Server transaction commits can be either fully durable, the SQL Server default, or delayed durable (also known as lazy commit).

Fully durable transaction commits are synchronous and report a commit as successful and return control to the client only after the log records for the transaction are written to disk. Delayed durable transaction commits are asynchronous and report a commit as successful before the log records for the transaction are written to disk. Writing the transaction log entries to disk is required for a transaction to be durable. Delayed durable transactions become durable when the transaction log entries are flushed to disk.

This topic details delayed durable transactions.

Full vs. Delayed Transaction Durability

Both full and delayed transaction durability have their advantages and disadvantages. An application can have a mix of fully and delayed durable transactions. You should carefully consider your business needs and how each fits into those needs.

Full transaction durability

Fully durable transactions write the transaction log to disk before returning control to the client. You should use fully durable transactions whenever:

- Your system cannot tolerate any data loss.
  See the section When can I lose data? for information on when you can lose some of your data.

- The bottleneck is not due to transaction log write latency.

  Delayed transaction durability reduces the latency due to log I/O by keeping the transaction log records in memory and writing to the transaction log in batches, thus requiring fewer I/O operations. Delayed transaction durability potentially reduces log I/O contention, thus reducing waits in the system.

  Full Transaction Durability Guarantees

- Once transaction commit succeeds, the changes made by the transaction are visible to the other transactions in the system. For more information about transaction isolation levels, see SET TRANSACTION ISOLATION LEVEL (Transact-SQL) or Transactions with Memory-Optimized Tables.

- Durability is guaranteed on commit. Corresponding log records are persisted to disk before the transaction commit succeeds and returns control to the client.

Delayed transaction durability

Delayed transaction durability is accomplished using asynchronous log writes to disk. Transaction log records are kept in a buffer and written to disk when the buffer fills or a buffer flushing event takes place. Delayed transaction durability reduces both latency and contention within the system because:

- The transaction commit processing does not wait for log IO to finish and return control to the client.

- Concurrent transactions are less likely to contend for log IO; instead, the log buffer can be flushed to disk in larger chunks, reducing contention, and increasing throughput.

NOTE

You may still have log I/O contention if there is a high degree of concurrency, particularly if you fill up the log buffer faster than you flush it.

When to use delayed transaction durability

Some of the cases in which you could benefit from using delayed transaction durability are:

You can tolerate some data loss.
If you can tolerate some data loss, for example, where individual records are not critical as long as you have most of the data, then delayed durability may be worth considering. If you cannot tolerate any data loss, do not use delayed transaction durability.

You are experiencing a bottleneck on transaction log writes.
If your performance issues are due to latency in transaction log writes, your application will likely benefit from using delayed transaction durability.

Your workloads have a high contention rate.
If your system has workloads with a high contention level much time is lost waiting for locks to be released. Delayed transaction durability reduces commit time and thus releases locks faster which results in higher throughput.

Delayed Transaction Durability Guarantees

- Once transaction commit succeeds, the changes made by the transaction are visible to the other transactions in the system.
- Transaction durability is guaranteed only following a flush of the in-memory transaction log to disk. The in-memory transaction log is flushed to disk when:
  - A fully durable transaction in the same database makes a change in the database and successfully commits.
  - The user executes the system stored procedure `sp_flush_log` successfully.

    If a fully durable transaction or sp_flush_log successfully commits, all previously committed delayed durability transactions are guaranteed to have been made durable.
  - SQL Server does attempt to flush the log to disk both based on log generation and on timing, even if all the transactions are delayed durable. This usually succeeds if the IO device is keeping up. However, SQL Server does not provide any hard durability guarantees other than durable transactions and sp_flush_log.

How to control transaction durability

Database level control

You, the DBA, can control whether users can use delayed transaction durability on a database with the following statement. You must set the delayed durability setting with ALTER DATABASE.

```
ALTER DATABASE … SET DELAYED_DURABILITY = { DISABLED | ALLOWED | FORCED }
```

DISABLED
[default] With this setting, all transactions that commit on the database are fully durable, regardless of the commit level setting (DELAYED_DURABILITY=[ON | OFF]). There is no need for stored procedure change and recompilation. This allows you to ensure that no data is ever put at risk by delayed durability.

ALLOWED
With this setting, each transaction's durability is determined at the transaction level – DELAYED_DURABILITY = { *OFF* | ON }. See Atomic block level control – Natively Compiled Stored Procedures and COMMIT level control – Transact-SQL for more information.

FORCED
With this setting, every transaction that commits on the database is delayed durable. Whether the transaction specifies fully durable (DELAYED_DURABILITY = OFF) or makes no specification, the transaction is delayed durable. This setting is useful when delayed transaction durability is useful for a database and you do not want to change any application code.

Atomic block level control – Natively Compiled Stored Procedures

The following code goes inside the atomic block.

```
DELAYED_DURABILITY = { OFF | ON }
```

OFF
[default] The transaction is fully durable, unless the database option DELAYED_DURABLITY = FORCED is in effect, in which case the commit is asynchronous and thus delayed durable. See Database level control for more information.

ON
The transaction is delayed durable, unless the database option DELAYED_DURABLITY = DISABLED is in effect, in which case the commit is synchronous and thus fully durable. See Database level control for more information.

Example Code:

```
CREATE PROCEDURE <procedureName> …
WITH NATIVE_COMPILATION, SCHEMABINDING, EXECUTE AS OWNER
AS BEGIN ATOMIC WITH
(
    DELAYED_DURABILITY = ON,
    TRANSACTION ISOLATION LEVEL = SNAPSHOT,
    LANGUAGE = N'English'
    …
)
END
```

Table 1: Durability in Atomic Blocks

ATOMIC BLOCK DURABILITY OPTION	NO EXISTING TRANSACTION	TRANSACTION IN PROCESS (FULLY OR DELAYED DURABLE)
DELAYED_DURABILITY = OFF	Atomic block starts a new fully durable transaction.	Atomic block creates a save point in the existing transaction, then begins the new transaction.
DELAYED_DURABILITY = ON	Atomic block starts a new delayed durable transaction.	Atomic block creates a save point in the existing transaction, then begins the new transaction.

COMMIT level control – Transact-SQL

The COMMIT syntax is extended so you can force delayed transaction durability. If DELAYED_DURABILITY is DISABLED or FORCED at the database level (see above) this COMMIT option is ignored.

```
COMMIT [ { TRAN | TRANSACTION } ] [ transaction_name | @tran_name_variable ] ] [ WITH ( DELAYED_DURABILITY = {
OFF | ON } ) ]
```

OFF
[default] The transaction COMMIT is fully durable, unless the database option DELAYED_DURABLITY = FORCED is in effect, in which case the COMMIT is asynchronous and thus delayed durable. See Database level control for more information.

ON
The transaction COMMIT is delayed durable, unless the database option DELAYED_DURABLITY = DISABLED is in effect, in which case the COMMIT is synchronous and thus fully durable. See Database level control for more information.

Summary of options and their interactions

This table summarizes the interactions between database level delayed durability settings and commit level settings. Database level settings always take precedence over commit level settings.

COMMIT SETTING/DATABASE SETTING	DELAYED_DURABILITY = DISABLED	DELAYED_DURABILITY = ALLOWED	DELAYED_DURABILITY = FORCED
DELAYED_DURABILITY = OFF Database level transactions.	Transaction is fully durable.	Transaction is fully durable.	Transaction is delayed durable.
DELAYED_DURABILITY = ON Database level transactions.	Transaction is fully durable.	Transaction is delayed durable.	Transaction is delayed durable.
DELAYED_DURABILITY = OFF Cross database or distributed transaction.	Transaction is fully durable.	Transaction is fully durable.	Transaction is fully durable.
DELAYED_DURABILITY = ON Cross database or distributed transaction.	Transaction is fully durable.	Transaction is fully durable.	Transaction is fully durable.

How to force a transaction log flush

There are two means to force flush the transaction log to disk.

- Execute any fully durable transaction that alters the same database. This forces a flush of the log records of all preceding committed delayed durability transactions to disk.
- Execute the system stored procedure `sp_flush_log`. This procedure forces a flush of the log records of all preceding committed delayed durable transactions to disk. For more information see sys.sp_flush_log (Transact-SQL).

Delayed durability and other SQL Server features

Change tracking and change data capture
All transactions with change tracking are fully durable. A transaction has the change tracking property if it does any write operations to tables that are enabled for change tracking. The use of delayed durability is not supported for databases which use change data capture (CDC).

Crash recovery

Consistency is guaranteed, but some changes from delayed durable transactions that have committed may be lost.

Cross-database and DTC
If a transaction is cross-database or distributed, it is fully durable, regardless of any database or transaction commit setting.

Always On Availability Groups and Mirroring
Delayed durable transactions do not guarantee any durability on either the primary or any of the secondaries. In addition, they do not guarantee any knowledge about the transaction at the secondary. After commit, control is returned to the client before any acknowledgement is received from any synchronous secondary. Replication to secondary replicas does continue to happen as flush to disk on the primary happens.

Failover clustering
Some delayed durable transaction writes might be lost.

Transaction Replication
Delayed durable transactions is not supported with Transactional Replication.

Log shipping
Only transactions that have been made durable are included in the log that is shipped.

Log Backup
Only transactions that have been made durable are included in the backup.

When can I lose data?

If you implement delayed durability on any of your tables, you should understand that certain circumstances can lead to data loss. If you cannot tolerate any data loss, you should not use delayed durability on your tables.

Catastrophic events

In the case of a catastrophic event, like a server crash, you will lose the data for all committed transactions that have not been saved to disk. Delayed durable transactions are saved to disk whenever a fully durable transaction is executed against any table (durable memory-optimized or disk-based) in the database, or `sp_flush_log` is called. If you are using delayed durable transactions, you may want to create a small table in the database that you can periodically update or periodically call `sp_flush_log` to save all outstanding committed transactions. The transaction log also flushes whenever it becomes full, but that is hard to predict and impossible to control.

SQL Server shutdown and restart

For delayed durability, there is no difference between an unexpected shutdown and an expected shutdown/restart of SQL Server. Like catastrophic events, you should plan for data loss. In a planned shutdown/restart some transactions that have not been written to disk may first be saved to disk, but you should not plan on it. Plan as though a shutdown/restart, whether planned or unplanned, loses the data the same as a catastrophic event.

See Also

Transactions with Memory-Optimized Tables

Database Checkpoints (SQL Server)

6/18/2018 • 8 minutes to read • Edit Online

THIS TOPIC APPLIES TO: ✓ SQL Server ✓ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

A *checkpoint* creates a known good point from which the SQL Server Database Engine can start applying changes contained in the log during recovery after an unexpected shutdown or crash.

Overview

For performance reasons, the Database Engine performs modifications to database pages in memory—in the buffer cache—and does not write these pages to disk after every change. Rather, the Database Engine periodically issues a checkpoint on each database. A *checkpoint* writes the current in-memory modified pages (known as *dirty pages*) and transaction log information from memory to disk and, also, records information about the transaction log.

The Database Engine supports several types of checkpoints: automatic, indirect, manual, and internal. The following table summarizes the types of **checkpoints:**

NAME	TRANSACT-SQL INTERFACE	DESCRIPTION
Automatic	EXEC sp_configure **'**recovery interval**','***seconds***'**	Issued automatically in the background to meet the upper time limit suggested by the **recovery interval** server configuration option. Automatic checkpoints run to completion. Automatic checkpoints are throttled based on the number of outstanding writes and whether the Database Engine detects an increase in write latency above 50 milliseconds. For more information, see Configure the recovery interval Server Configuration Option.
Indirect	ALTER DATABASE … SET TARGET_RECOVERY_TIME =*target_recovery_time* { SECONDS | MINUTES }	Issued in the background to meet a user-specified target recovery time for a given database. Beginning with SQL Server 2016 (13.x), the default value is 1 minute. The default is 0 for older versions, which indicates that the database will use automatic checkpoints, whose frequency depends on the recovery interval setting of the server instance. For more information, see Change the Target Recovery Time of a Database (SQL Server).

NAME	TRANSACT-SQL INTERFACE	DESCRIPTION
Manual	CHECKPOINT [*checkpoint_duration*]	Issued when you execute a Transact-SQL CHECKPOINT command. The manual checkpoint occurs in the current database for your connection. By default, manual checkpoints run to completion. Throttling works the same way as for automatic checkpoints. Optionally, the *checkpoint_duration* parameter specifies a requested amount of time, in seconds, for the checkpoint to complete. For more information, see CHECKPOINT (Transact-SQL).
Internal	None.	Issued by various server operations such as backup and database-snapshot creation to guarantee that disk images match the current state of the log.

NOTE

The **-k** SQL Server advanced setup option enables a database administrator to throttle checkpoint I/O behavior based on the throughput of the I/O subsystem for some types of checkpoints. The **-k** setup option applies to automatic checkpoints and any otherwise unthrottled manual and internal checkpoints.

For automatic, manual, and internal checkpoints, only modifications made after the latest checkpoint need to be rolled forward during database recovery. This reduces the time required to recover a database.

IMPORTANT

Long-running, uncommitted transactions increase recovery time for all checkpoint types.

Interaction of the TARGET_RECOVERY_TIME and 'recovery interval' Options

The following table summarizes the interaction between the server-wide **sp_configure'**recovery interval' setting and the database-specific ALTER DATABASE ... TARGET_RECOVERY_TIME setting.

TARGET_RECOVERY_TIME	'RECOVERY INTERVAL'	TYPE OF CHECKPOINT USED
0	0	automatic checkpoints whose target recovery interval is 1 minute.
0	>0	Automatic checkpoints whose target recovery interval is specified by the user defined setting of the **sp_configurerecovery interval** option.

TARGET_RECOVERY_TIME	'RECOVERY INTERVAL'	TYPE OF CHECKPOINT USED
>0	Not applicable.	Indirect checkpoints whose target recovery time is determined by the TARGET_RECOVERY_TIME setting, expressed in seconds.

Automatic checkpoints

An automatic checkpoint occurs each time the number of log records reaches the number the Database Engine estimates it can process during the time specified in the **recovery interval** server configuration option. For more information, see Configure the recovery interval Server Configuration Option.

In every database without a user-defined target recovery time, the Database Engine generates automatic checkpoints. The frequency depends on the **recovery interval** advanced server configuration option, which specifies the maximum time that a given server instance should use to recover a database during a system restart. The Database Engine estimates the maximum number of log records it can process within the recovery interval. When a database using automatic checkpoints reaches this maximum number of log records, the Database Engine issues an checkpoint on the database.

The time interval between automatic checkpoints can be **highly** variable. A database with a substantial transaction workload will have more frequent checkpoints than a database used primarily for read-only operations. Under the simple recovery model, an automatic checkpoint is also queued if the log becomes 70 percent full.

Under the simple recovery model, unless some factor is delaying log truncation, an automatic checkpoint truncates the unused section of the transaction log. By contrast, under the full and bulk-logged recovery models, once a log backup chain has been established, automatic checkpoints do not cause log truncation. For more information, see The Transaction Log (SQL Server).

After a system crash, the length of time required to recover a given database depends largely on the amount of random I/O needed to redo pages that were dirty at the time of the crash. This means that the **recovery interval** setting is unreliable. It cannot determine an accurate recovery duration. Furthermore, when an automatic checkpoint is in progress, the general I/O activity for data increases significantly and quite unpredictably.

Impact of recovery interval on recovery performance

For an online transaction processing (OLTP) system using short transactions, **recovery interval** is the primary factor determining recovery time. However, the **recovery interval** option does not affect the time required to undo a long-running transaction. Recovery of a database with a long-running transaction can take much longer than the specified in the **recovery interval** option.

For example, if a long-running transaction took two hours to perform updates before the server instance became disabled, the actual recovery takes considerably longer than the **recovery interval** value to recover the long transaction. For more information about the impact of a long running transaction on recovery time, see The Transaction Log (SQL Server).

Typically, the default values provides optimal recovery performance. However, changing the recovery interval might improve performance in the following circumstances:

- If recovery routinely takes significantly longer than 1 minute when long-running transactions are not being rolled back.
- If you notice that frequent checkpoints are impairing performance on a database.

If you decide to increase the **recovery interval** setting, we recommend increasing it gradually by small increments and evaluating the effect of each incremental increase on recovery performance. This approach is important because as the **recovery interval** setting increases, database recovery takes that many times longer to complete. For example, if you change **recovery interval** 10 minutes, recovery takes approximately 10 times longer to complete than when **recovery interval** is set to 1 minute.

Indirect checkpoints

Indirect checkpoints, introduced in SQL Server 2012 (11.x), provide a configurable database-level alternative to automatic checkpoints. This can be configured by specifying the **target recovery time** database configuration option. For more information, see Change the Target Recovery Time of a Database (SQL Server). In the event of a system crash, indirect checkpoints provide potentially faster, more predictable recovery time than automatic checkpoints. Indirect checkpoints offer the following advantages:

- An online transactional workload on a database configured for indirect checkpoints can experience performance degradation. Indirect checkpoints ensure that the number of dirty pages are below a certain threshold so the database recovery completes within the target recovery time.

  The **recovery interval** configuration option uses the number of transactions to determine the recovery time, as opposed to **indirect checkpoints** which makes use of number of dirty pages. When indirect checkpoints are enabled on a database receiving a large number of DML operations, the background writer can start aggressively flushing dirty buffers to disk to ensure that the time required to perform recovery is within the target recovery time set of the database. This can cause additional I/O activity on certain systems which can contribute to a performance bottleneck if the disk subsystem is operating above or nearing the I/O threshold.

- Indirect checkpoints enable you to reliably control database recovery time by factoring in the cost of random I/O during REDO. This enables a server instance to stay within an upper-bound on recovery times for a given database (except when a long-running transaction causes excessive UNDO times).

- Indirect checkpoints reduce checkpoint-related I/O spiking by continually writing dirty pages to disk in the background.

However, an online transactional workload on a database configured for indirect checkpoints can experience performance degradation. This is because the background writer used by indirect checkpoint sometimes increases the total write load for a server instance.

IMPORTANT

Indirect checkpoint is the default behavior for new databases created in SQL Server 2016 (13.x), including the Model and TempDB databases.
Databases that were upgraded in-place, or restored from a previous version of SQL Server, will use the previous automatic checkpoint behavior unless explicitly altered to use indirect checkpoint.

Internal checkpoints

Internal Checkpoints are generated by various server components to guarantee that disk images match the current state of the log. Internal checkpoint are generated in response to the following events:

- Database files have been added or removed by using ALTER DATABASE.

- A database backup is taken.

- A database snapshot is created, whether explicitly or internally for DBCC CHECK.

- An activity requiring a database shutdown is performed. For example, AUTO_CLOSE is ON and the last user connection to the database is closed, or a database option change is made that requires a restart of the database.

- An instance of SQL Server is stopped by stopping the SQL Server (MSSQLSERVER) service . Either action causes a checkpoint in each database in the instance of SQL Server.
- Bringing a SQL Server failover cluster instance (FCI) offline.

Related tasks

To change the recovery interval on a server instance

- Configure the recovery interval Server Configuration Option

  To configure indirect checkpoints on a database

- Change the Target Recovery Time of a Database (SQL Server)

  To issue a manual checkpoint on a database

- CHECKPOINT (Transact-SQL)

See also

The Transaction Log (SQL Server)
Transaction Log Physical Architecture (From SQL Server 2008 R2 Books Online but still applicable!)

Change the Target Recovery Time of a Database (SQL Server)

5/3/2018 • 2 minutes to read • Edit Online

THIS TOPIC APPLIES TO: ✓ SQL Server ✗ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

This topic describes how to set the change the target recovery time of a SQL Server database in SQL Server 2017 by using SQL Server Management Studio or Transact-SQL. By default, the target recovery time is 60 seconds, and the database uses *indirect checkpoints*. The target recovery time establishes an upper-bound on recovery time for this database.

NOTE

The upper-bound that is specified for a given database by its target recovery time setting could be exceeded if a long-running transaction causes excessive UNDO times.

- **Before you begin:** Limitations and Restrictions, Security
- **To change the target recovery time, using:** SQL Server Management Studio or Transact-SQL

Before You Begin

Limitations and Restrictions

Caution

An online transactional workload on a database that is configured for indirect checkpoints could experience performance degradation. Indirect checkpoints make sure that the number of dirty pages are below a certain threshold so that the database recovery completes within the target recovery time. The recovery interval configuration option uses the number of transactions to determine the recovery time as opposed to indirect checkpoints which makes use of number of dirty pages. When indirect checkpoints are enabled on a database receiving a large number of DML operations, the background writer can start aggressively flushing dirty buffers to disk to ensure that the time required to perform recovery is within the target recovery time set of the database. This can cause additional I/O activity on certain systems which can contribute to a performance bottleneck if the disk subsystem is operating above or nearing the I/O threshold.

Security

Permissions

Requires ALTER permission on the database.

Using SQL Server Management Studio

To change the target recovery time

1. In **Object Explorer**, connect to an instance of the SQL Server Database Engine, and expand that instance.
2. Right-click the database you want to change, and click the **Properties** command.
3. In the **Database Properties** dialog box, click the **Options** page.
4. In the **Recovery** panel, in the **Target Recovery Time (Seconds)** field, specify the number of seconds that you want as the upper-bound on the recovery time for this database.

Using Transact-SQL

To change the target recovery time

1. Connect to the instance of SQL Server where the database resides.

2. Use the following ALTER DATABASEstatement, as follows:

   TARGET_RECOVERY_TIME =*target_recovery_time* { SECONDS | MINUTES }

   target_recovery_time
   Beginning with SQL Server 2016 (13.x), the default value is 1 minute. When greater than 0 (the default for older versions), specifies the upper-bound on the recovery time for the specified database in the event of a crash.

   SECONDS
   Indicates that *target_recovery_time* is expressed as the number of seconds.

   MINUTES
   Indicates that *target_recovery_time* is expressed as the number of minutes.

   The following example sets the target recovery time of the **AdventureWorks2012** database to `60` seconds.

   ```
   ALTER DATABASE AdventureWorks2012 SET TARGET_RECOVERY_TIME = 60 SECONDS;
   ```

See Also

Database Checkpoints (SQL Server)
ALTER DATABASE SET Options (Transact-SQL)