

Other errors and exceptions

Last updated by | Vitor Tomaz | Jun 8, 2022 at 5:37 AM PDT

Contents

- See [Other errors and exceptions under SQL DB as well for ...](#)
- Self-help content presented in Azure Portal
 - Error 9002: The transaction log for database X is full
 - Error 701: There is insufficient system memory to run this ...
 - Common error messages and solutions
 - Error 10930: The service is currently too busy
 - Error 845: Time-out occurred while waiting for buffer latch ...
 - Performance degradation due to IO latency
 - Identity field jump - gap in incremental sequence
 - IDENTITY_CACHE is on
 - Reuse of values
 - Monitor and performance tuning
 - Resources
 - Performance recommendations if migrating to Managed I...
 - Instance settings on SQL Server and Managed Instance for...
 - Resources

See [Other errors and exceptions](#) under SQL DB as well for more TSGs

Self-help content presented in Azure Portal

(This content was shown to the customer during case submission. It's also visible on 'Diagnose and solve problems' blade.)

Error 9002: The transaction log for database X is full

Errors 9002 or 40552 might appear if the transaction log is full and doesn't accept new transactions. These errors occur when the database transaction log, managed by Azure SQL Managed Instance, exceeds space thresholds.

Like SQL Server, the transaction log for each database is truncated if a log backup occurs. Truncation leaves empty space in the log file, which can access new transactions. When the log file can't be truncated by log backups, the log file grows to accommodate new transactions. If the log file grows to its maximum limits, new transactions aren't accepted.

To see what is preventing log truncation, refer to `log_reuse_wait_desc` in `sys.databases`.

```
SELECT [name], log_reuse_wait_desc FROM sys.databases;
```

The log reuse wait provides conditions or causes preventing the transaction log from being truncated by a regular log backup. For more information, see [sys.databases \(Transact-SQL\)](#).

Learn how to avoid this in the future: [Troubleshooting transaction log errors with Azure SQL Database and Azure SQL Managed Instance](#).

Error 701: There is insufficient system memory to run this query

SQL Server failed to allocate sufficient memory to run the query. This can be caused by:

- Operating system settings
- Physical memory availability
- Memory limits on a workload

In most cases, the failed transaction is not the cause of this error. Use [MSSQLSERVER 701](#) for assistance with troubleshooting the error.

Common error messages and solutions

Error 10930: The service is currently too busy

Use [Troubleshooting network performance](#) to standardize and consistently test network latency and bandwidth between two hosts. Observe the Azure network to help isolate issues.

Error 845: Time-out occurred while waiting for buffer latch type X for page Y database ID Z

Use [Troubleshooting network performance](#).

"A process waited to acquire a latch, but the process waited until the time limit expired and failed to acquire one." This occurs if an I/O operation takes too long to complete, usually because other tasks blocked system processes.

Performance degradation due to IO latency

In the General Purpose service tier, database files get dedicated IOPS and throughput depending on the file size. Larger files get more IOPS and throughput. See [File IO characteristics in General Purpose tier](#).


See [Increase data file size to improve HammerDB workload performance on General Purpose Managed Instance](#).

There is an instance-level limit on the maximum log-write throughput. You might not be able to reach the maximum file throughput on the log file because the instance throughput limit was reached.



When the IOPS or throughput generated by the database workload exceeds the limits of a database file/blob, storage throttling occurs. See [Premium SSDs](#).

For MI GP instances, a typical storage throttling symptom is high IO latency. IO latency can rise to hundreds of milliseconds while being throttled. Without throttling, average storage latency at the OS level is about 2-10

milliseconds. Another symptom of storage throttling is long PAGEIOLATCH waits (for data file IO) and WRITELOG waits (for the transaction log file IO). During storage throttling, there is a negative effect on database performance.

If there are symptoms of storage throttling while running workloads on MI GP, increase database file size to get more IOPS/throughput from Azure Premium Storage. You can use the script [managed-instance](#)  to determine if storage IO generated by the database workload approaches Azure Premium Storage IOPS/throughput limits.

Resources

- [Storage performance best practices and considerations for Azure SQL DB Managed Instance \(General Purpose\)](#) 
- [Overview of Azure SQL Managed Instance resource limits](#) 

Identity field jump - gap in incremental sequence

The identity field value might increase causing a gap to occur in the incremental sequence. Consider the following guidance.

IDENTITY_CACHE is on

The behavior occurs because SQL Server generated and cached the table identity values. This improves performance of the INSERT statements, which run on the table. In some situations, the pre-generated identity value is lost, and that behavior is responsible for the value increase in the identity.

Run the following query to identify if the value is on:

```
select *  
from sys.database_scoped_configurations  
where [name] = N'IDENTITY_CACHE'
```

If the value is on, run the following query to turn off the IDENTITY_CACHE:


```
ALTER DATABASE SCOPED CONFIGURATION SET IDENTITY_CACHE = OFF
```

Additional references: [ALTER DATABASE SCOPED CONFIGURATION \(Transact-SQL\)](#) .




Reuse of values


For an identity property with seed/increment, the engine doesn't reuse identity values. If an insert statement fails or if the insert statement is rolled back, then the consumed identity values are lost and not generated again. This can cause gaps when the subsequent identity values are generated.

Reference: [CREATE TABLE \(Transact-SQL\) IDENTITY \(Property\)](#) .


To resolve this, you can record a profiler trace. Use [SQL Server Profiler](#)  to help locate the failed transaction. After the transaction is identified, update the code that was inserting an incorrect ID.


Monitor and performance tuning

SQL Server has a monitoring and diagnostic capabilities that SQL Database and SQL Managed Instance use, such as [query store](#)  and [dynamic management views \(DMVs\)](#) . See [Monitoring using DMVs](#)  for scripts to monitor performance.

Query Performance Insight doesn't support SQL Database Managed Instance, and there is no delivery date for when it will. Instead, use [Azure SQL Analytics](#)  to monitor Managed Instance performance, with AI troubleshooting.



Resources

Here is a [Short demo video](#)  that walks through Performance Monitoring for Azure SQL Managed Instance with Azure SQL Analytics.

In addition, you can use the [JocaPC/qpi](#)  library to track performance of workloads in Managed Instance using TSQL.

Performance recommendations if migrating to Managed Instance

If you migrate your databases from SQL Server (on-premises or Azure VM) to Azure SQL Managed Instance, compare the performance of your database in the new environment with the performance of the original database on the source SQL Server. Performance of the databases on Managed Instance might be worse than the performance of the source SQL Server. This is sometimes expected because Managed Instance has some management overhead (automatic backups, a guarantee of 99.99% availability), while in other cases, these are some of the settings that can be configured to improve performance of Managed Instance.

If there are performance differences between Managed Instance and SQL server, see this [medium.com](#)  article: [How to identify why workload performance on Azure SQL Managed Instance are different than SQL Server](#) .

Instance settings on SQL Server and Managed Instance for optimal performance

There can be performance differences between SQL Server and Managed Instance, because their databases aren't configured in the same way:

- Different recovery model
- Compatibility level
- Legacy cardinality estimator
- Trace flags
- Encryption
- Tempdb settings

If you migrate your databases from one SQL Server instance to another, or from SQL Server to cloud, (for example, Azure SQL Managed Instance), compare the workload performance between source and target environment. Sometimes there is a difference in performance, even though the source and target environment seem the same.

Factors that can cause differing performance on source and target instances:

- Server or database properties on source and target instance:
 - Compatibility levels
 - Cardinality estimator

- Encryption
- Trace flag settings
- Tempdb settings: number of files, encryption

If there is differing performance, compare the settings of the SQL Server and Managed Instance on which the performance tests are conducted. SQL scripts are in GitHub, see [sql-server-samples](#).

See also the [medium.com](#) article [Compare environment settings on SQL Server and Azure SQL that may impact performance](#).

Resources

- [Troubleshoot transient connection errors in SQL Database and SQL Managed Instance](#)
- [Troubleshooting connectivity issues and other errors with Azure SQL Database and Azure SQL Managed Instance](#)
- [Database engine errors](#)

How good have you found this content?



-