

Configure databases for optimal performance

1. Introduction

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

Introduction

Completed

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

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

Learning objectives

In this module, you will:

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

2. Explore database maintenance checks

Explore database maintenance checks

Completed

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

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

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

Rebuild and reorganize

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

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

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

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

The common guidance for index maintenance is:

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

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

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

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

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

Statistics

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

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

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

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

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

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

Create statistics

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

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

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

Whenever you encounter the following situations, consider creating statistics:

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

Maintenance tasks automation

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

SQL Server on an Azure Virtual Machine

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

Azure SQL Database

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

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

Azure SQL Managed Instance

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

3. Describe database scoped configuration options

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

Describe database scoped configuration options

Completed

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

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

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

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

The above settings are a subset of the configurable options.

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

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

Database compatibility level

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

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

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

4. Describe automatic tuning

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

Describe automatic tuning

Completed

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

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

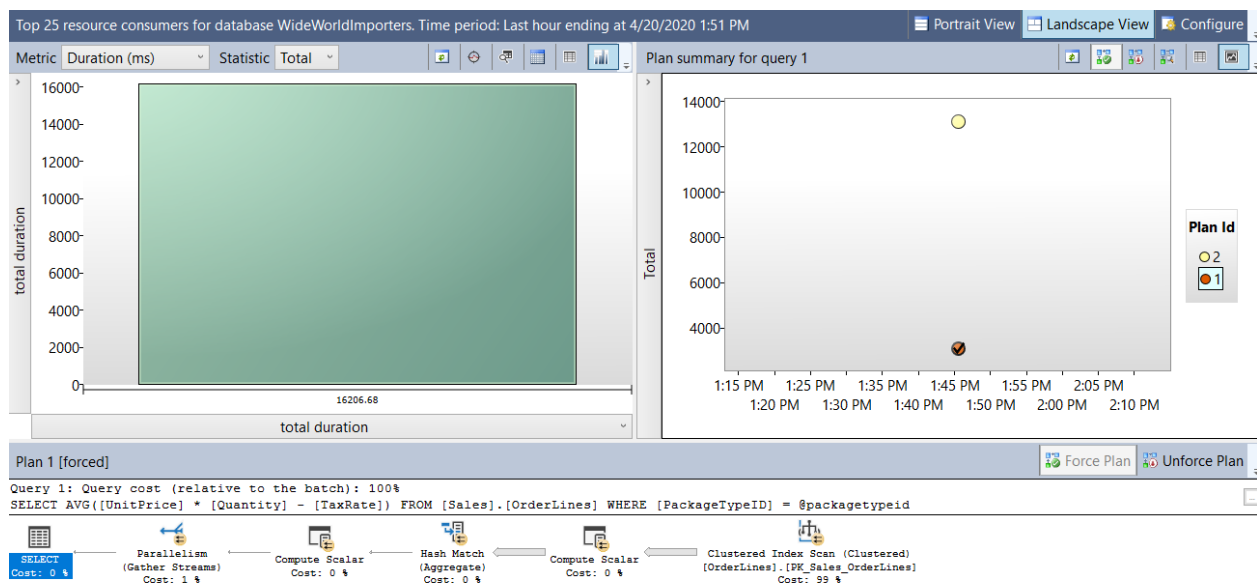
cause updates to the statistics. This evolution can cause queries to perform poorly as the execution plan no longer meets the demands of the given query.

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

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

Automatic plan correction

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



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

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

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

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

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

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

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

Automatic index management

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

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

Inherit from: ⓘ

Server Azure defaults Don't inherit

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

Configure the automatic tuning options ⓘ

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

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

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

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

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

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

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

5. Describe intelligent query processing

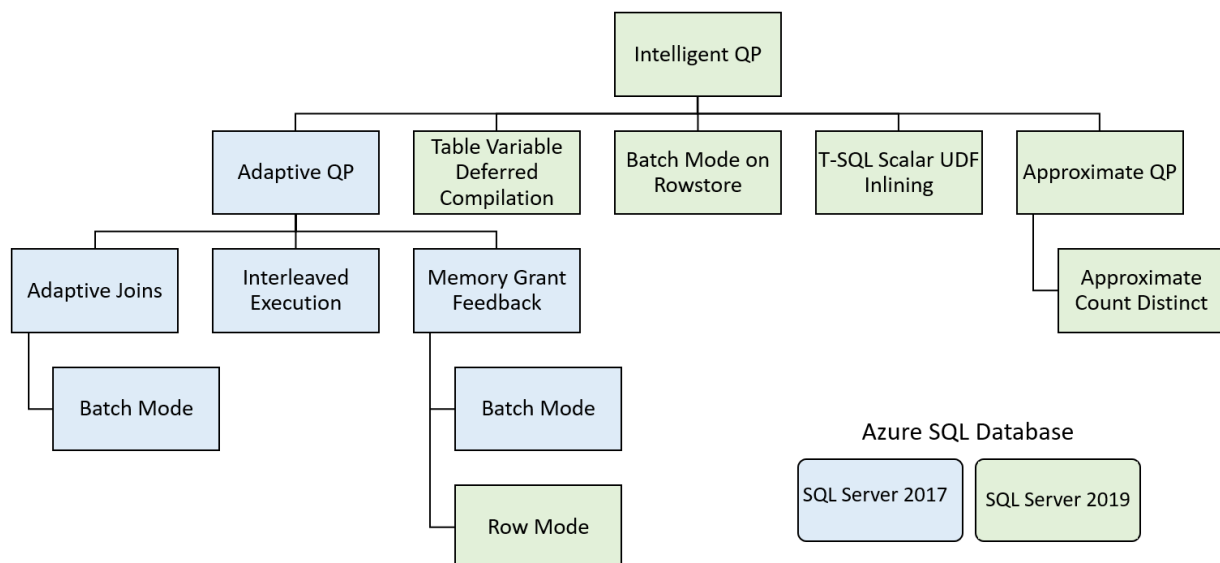
<https://learn.microsoft.com/en-us/training/modules/configure-databases-for-optimal-performance/5-describe-intelligent-query-processing>

Describe intelligent query processing

Completed

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

These features break down into a few families of features:



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

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

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

Adaptive query processing

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

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

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

Table variable deferred compilation

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

Batch mode on row store

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

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

Scalar user-defined function inlining

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

Approximate count distinct

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

6. Exercise: Detect and correct fragmentation issues

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

Exercise: Detect and correct fragmentation issues

Completed

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

Note

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

Launch the exercise and follow the instructions.

[Launch Exercise](#)

7. Module assessment

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

Module assessment

Completed

8. Summary

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

Summary

Completed

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

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