



Intelligent Performance Optimization with Microsoft Azure SQL Database

A cloud developer's guide to
resource scaling, performance
features, and automatic tuning

Performance is critical no matter what your data does. Just as a dropped call frustrates cellular customers, an app with data issues—out-of-date records, latency, disconnects, or worse—will drive your users to your competitors. This e-book provides the technical details necessary to help ensure that your data won't let you down when you need it the most.

Who should read this?

We wrote this e-book for developers concerned with how their database performance supports their apps. We cover the many ways that Microsoft Azure SQL Database helps you deliver great performance, at minimal overhead. We capture the deep technical details, like the relevant SQL code snippets, and share the process for putting each feature or capability to use. If you want cloud database performance that stands up to every development scenario, read on.

© 2018 Microsoft Corporation. All rights reserved.

This document is provided "as is." Information and views expressed in this document, including URL and other internet website references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

Contents

1	Introduction
3	Dynamic resource scaling
3	Service tiers, performance levels, and DTUs
5	Elastic database pools
7	Database performance features
7	In-Memory OLTP
11	Columnstore indexes
14	Intelligent performance monitoring and tuning
14	Intelligent Insights
15	Automatic tuning
17	Adaptive query processing
19	Conclusion

Introduction

No matter what your app does, making sure it delivers adequate performance is essential.

If it doesn't, one thing's for sure: you'll be hearing about it from someone, and you'll need to scramble to react. In many cases, the cause of such problems can be your database. It may be starved for resources, or the issue could be something else. Either way, you'd probably rather not have to worry about it. But what can you do to avoid such problems?

Azure SQL Database, the intelligent cloud database from Microsoft, provides several ways to ensure you'll always have the database performance you need. This e-book examines Azure SQL Database performance from three perspectives, including the features and capabilities you can put to use in each area to optimize database performance:

Dynamic resource scaling.

With Azure SQL Database, each single database you create has a guaranteed set of dedicated resources, which you can scale up or down at any time. You can also group databases into elastic database pools that share a common set of dedicated resources, enabling you to maximize resource usage and save money while ensuring you'll always be able to scale if and when it's needed.

Database performance features.

Before paying for more database resources, you'll want to make sure you're getting the most out of the resources you already have. Azure SQL Database includes powerful in-memory technologies designed to help you do just that, regardless of whether you're running an online transaction processing (OLTP) or analytics workload. These capabilities are straightforward to use, are proven to improve performance by up to two orders of magnitude, and can be turned on with just a few lines of code.

Intelligent performance monitoring and tuning.

Achieving top performance isn't just about scalability and resources; it's also about learning and optimizing. As an intelligent database, Azure SQL Database is designed to do the heavy lifting in this area for you. Automatic performance monitoring and tuning is available across service tiers and performance levels, ready to work on your behalf in the background to make sure you get the most for your money regardless of your database size, structure, or workload.



When it comes to performance, Azure SQL Database is built for you, the developer. Azure SQL Database is designed to make your work easier and let you focus on other priorities. It scales on the fly, without downtime, with capabilities that make it great for building multitenant software as a service (SaaS) apps. Plus, it learns and adapts to your database workload, whatever that may be, using built-in intelligence to continually adjust and improve over time.

That said, there's more to a successful app than impressive performance. That's why we built Azure SQL Database to be just as intelligent in other areas, such as availability and security, so that you won't need to worry about those essentials either. Finally, we built Azure SQL Database to let you work the way *you* want, using the tools and platforms you prefer. It's why we call Azure SQL Database the intelligent database *for developers*. Read on to learn more, and we're sure you'll agree.

Dynamic resource scaling

Azure SQL Database delivers predictable, dynamically scalable performance—without downtime.

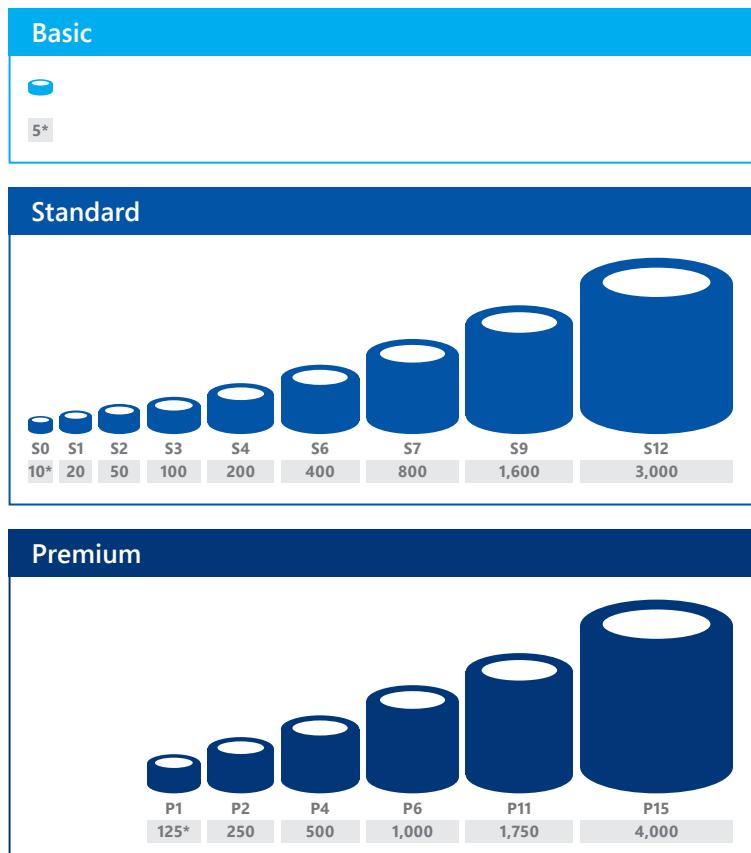
Each single database you create is portable and isolated from all others, with a guaranteed set of dedicated resources (called Database Transaction Units, or DTUs) based on the service tier and performance level you select for that database. You can also group databases into elastic database pools that share a common set of dedicated resources (called elastic DTUs, or eDTUs), enabling you to maximize resource usage and save money while ensuring you'll always be able to scale if and when it's needed.

Many of the other performance features of Azure SQL Database build on those concepts, such as being available only within certain service tiers. So it's worth examining each of the concepts a bit more closely before moving on to other ways Azure SQL Database can help you achieve the performance you need.

Service tiers, performance levels, and DTUs

Azure SQL Database provides a wide range of performance levels—from 5 DTUs to 4,000 DTUs—across three service tiers: basic,

standard, and premium. The premium tier is best suited to input/output (I/O)-intensive workloads, providing an order of magnitude more throughput-per-DTU and lower latency-per-I/O than the standard and basic tiers. Each performance level includes a specified amount of storage; additional storage can also be purchased.



* Database Transaction Units (DTUs)

But what does this mean to you as a developer? It enables you to start small and dynamically scale, paying only for the resources you need, when you need them—without having to worry about being able to handle future growth, or paying for the ability to do so before it's needed.

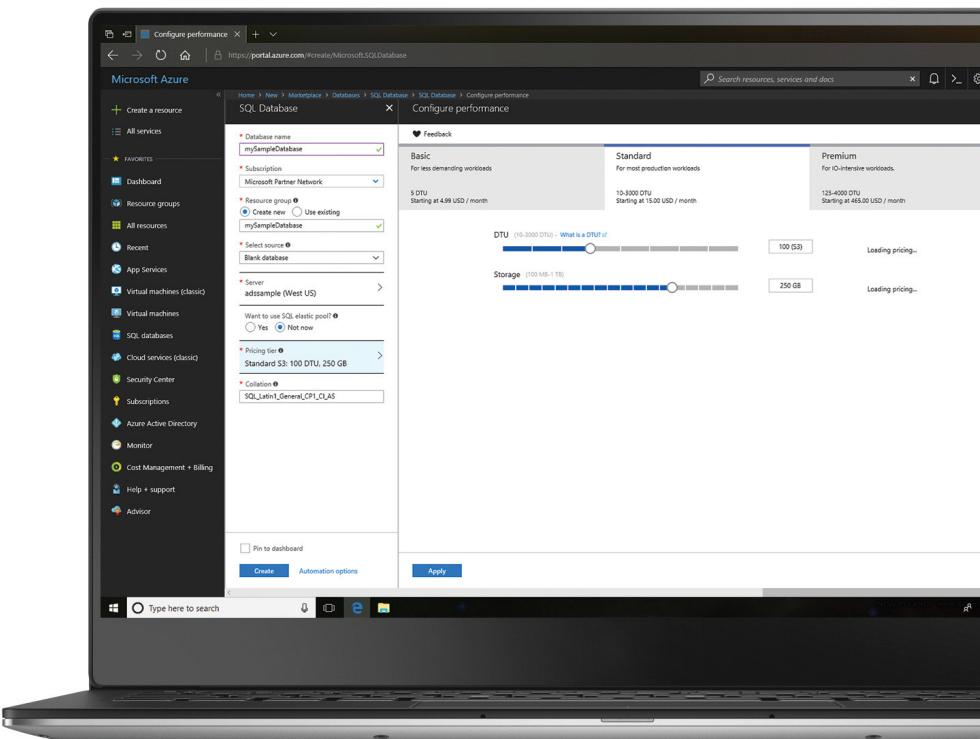
Here are some resources for learning more about service tiers, performance levels, and related concepts:

- Get an [overview of the service tiers](#), including how to choose one and the maximum resources available within each tier.
- Find out about [resource limits](#) for each performance level within each service tier.
- Learn more about [DTUs and eDTUs](#), including how these metrics guarantee a certain level of resources for your single database or elastic database pool, how to determine the DTUs needed by your workload, and what happens when you hit your maximum DTUs.

Putting Azure SQL Database service tiers to use

You can set or change the service tier, performance level, or storage amount for a new or existing single database using the Azure portal, as shown below. It's as easy as selecting **Pricing tier (scale DTUs)** to open the **Configure performance** page for your database, after which you can:

- Set or change the service tier for your workload.
- Set or change the performance level (DTUs) within a service tier using the **DTU** slider.
- Set or change the storage amount for the performance level using the **Storage** slider.



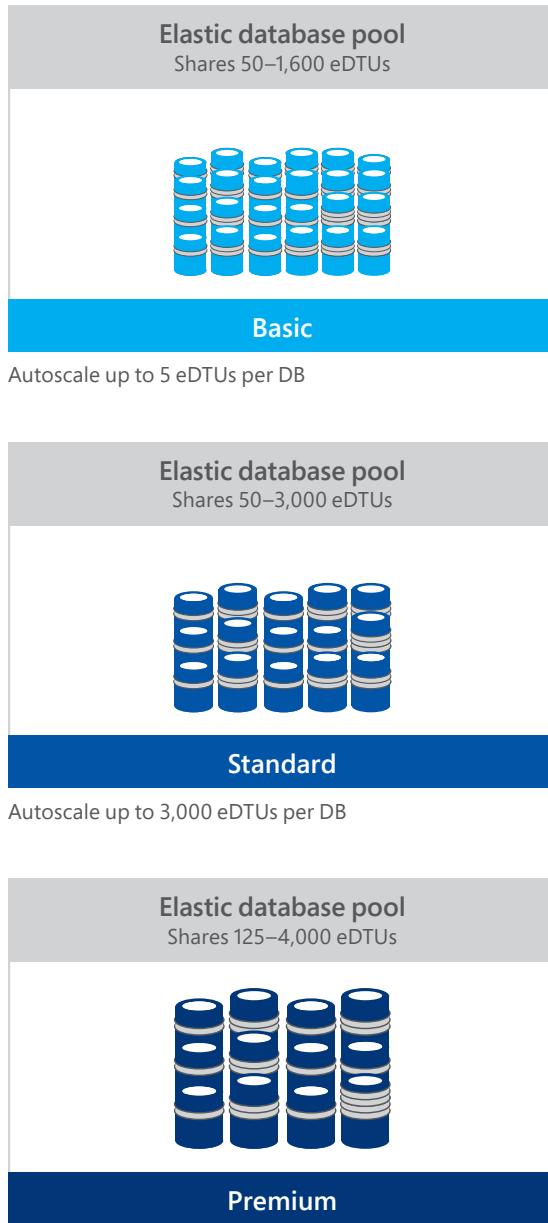
See how to [manage resources for a single database](#) using the Azure portal, Azure PowerShell, Azure CLI, Transact-SQL (T-SQL), and the REST API. You can even configure your database to [scale automatically](#).

Elastic database pools

The ability to dial performance up or down for a single database on demand is great for many situations, especially if database usage is relatively predictable in the short term—such as when you’re building a line-of-business app. But what if usage patterns are varied and unpredictable, and what if you have many such usage patterns to consider? For example, what if you have a multitenant SaaS app with a single database per tenant? How can you ensure reliable performance for all your customers *and* minimize your own database costs, without overprovisioning each customer’s database and paying for spare capacity that, in many cases, will rarely or never be used?

Elastic database pools, a feature in Azure SQL Database, are designed to solve this very problem, making them a powerful tool for building modern, multitenant SaaS apps. The concept is simple: you group individual databases into an elastic database pool, and then allocate performance resources to that pool. The individual databases within the pool can then draw upon those resources up to the limits you set. Setting up an elastic database pool is like selecting a service tier and performance level for a single database. There are a few more parameters you’ll need to set for how resources are shared, but that’s about it.

With elastic database pools, you won’t need to focus on dialing the performance of individual





databases up or down. Just keep an eye on the overall resource usage within the pool, and dial the eDTUs for the pool up or down as needed, such as when you add new customers or deliver new features that increase per-customer usage. You can control the minimum and maximum resources that are available to each database in the pool to ensure that no one database uses all the resources and that every database has a guaranteed minimum set of resources. The end result: your customers get the performance they expect, but you won't need to waste money on spare performance capacity. And as an added benefit, you'll be able to use [elastic database jobs](#) to manage all those databases as if they were one.

Putting elastic database pools to use

Learn more about [elastic database pools](#), such as:

- When to consider using them.
- How to choose the correct pool size.
- How to manage them using the Azure portal, Azure PowerShell, Azure CLI, T-SQL, and the REST API.

This [Channel 9 video](#) provides an overview of elastic database pools and elastic database jobs, including demos. Also, you may want check out the previously mentioned articles on [service tiers](#), [resource limits](#), and [DTUs/eDTUs](#)—they all apply to elastic database pools as well as single databases. Finally, if you're planning to build a multitenant SaaS app, read this overview of the pros and cons of different tenancy models.

Finally, if you're planning to build a multitenant SaaS app, read this [overview](#) of the pros and cons of different tenancy models.

Database performance features

Before you pay more for a higher service tier or performance level, you'll want to know that you're getting the most from the resources you already have. In-Memory OLTP and columnstore indexes, two built-in performance features of Azure SQL Database, can help you do that.

For example, with these features, you could support your workload with a P2 performance level; without them, that same workload might require a P6.

In-Memory OLTP improves the performance of high-throughput transaction processing, data ingestion, and transient data scenarios.

Columnstore indexes improve the performance of analytical queries.

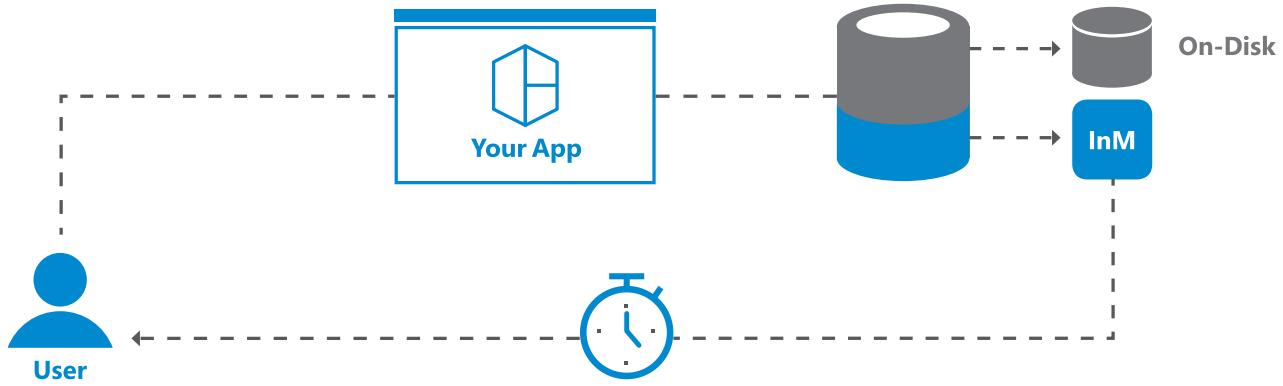
In-Memory OLTP

How do you address the performance limitations caused by CPU and concurrency bottlenecks? For example, say you have a web app that, for security reasons, logs every request and response into an audit table. The more the app is used, the more database traffic hits this table. If traffic grows too much, it

could degrade the entire app's performance. Similarly, you could be getting bogged down by unnecessary log I/O when working with temporary tables and table variables.

Sure, you could try to address performance issues by rearchitecting your app—for example, by using asynchronous functions to unblock database calls. Or you could decouple application components using stateless microservices, relying on eventual data concurrency. But what if there was a faster and easier way to improve performance, without changing the app itself or increasing the resources allocated to the database?

In-Memory OLTP, a feature of Azure SQL Database (and Microsoft SQL Server), removes concurrency bottlenecks and makes data access more efficient to improve performance by a factor of up to 30. It's useful for high-throughput transaction processing (such as trading and gaming), data ingestion from events or Internet of Things (IoT) devices, caching, data load, and temporary table and table variable scenarios. And even though the feature has "OLTP" in its name, that doesn't mean you can't use it for certain aspects of online analytical processing (OLAP) workloads or mixed hybrid transaction/analytical processing (HTAP) workloads, such as for



staging or temporary tables. (For example, you can use In-Memory OLTP to completely eliminate log I/O for operations involving temporary tables and table variables.)

In essence, In-Memory OLTP is a memory-optimized database engine integrated into Azure SQL Database. It improves performance by optimizing the efficiency of data access and transaction execution and by removing lock and latch contention between concurrently executing transactions. Put another way, it's not fast just because the data is in-memory; it's fast because it's optimized for data being in-memory.

In-Memory OLTP is also robust—you won't lose your data if there's a failure. By default, all transactions are fully durable, meaning that you have the same durability guarantees you get for any other table in Azure SQL Database. When the transaction commits, all changes are written to the transaction log on disk. If there's a failure at any time after the transaction commits, your data is there when the database

comes back online. In addition, In-Memory OLTP works with all high-availability and disaster-recovery capabilities of Azure SQL Database, like point-in-time restore, geo-restore, and active geo-replication.

In-Memory OLTP and other in-memory features—such as columnstore indexes (discussed in the next section)—are available in all databases in the premium tier of Azure SQL Database, including those in premium elastic database pools.

Putting In-Memory OLTP to use

You can take advantage of In-Memory OLTP by using one or more of the following types of objects:

Memory-optimized tables, which store user data. You declare a table to be memory optimized at the time the table is created.

Nondurable tables, which handle transient data, such as for caching or for intermediate result sets (replacing traditional temp

tables). A nondurable table is a memory-optimized table that's declared with DURABILITY=SCHEMA_ONLY, meaning that changes to these tables don't incur any I/O. This avoids consuming log I/O resources for cases where durability isn't a concern.

Memory-optimized table types, which are used for table-valued parameters (TVPs), plus intermediate result sets in stored procedures. These can be used instead of traditional table types. Table variables and TVPs that are declared using a memory-optimized table type inherit the benefits of nondurable memory-optimized tables: efficient data access and no I/O.

Natively compiled T-SQL modules, which can further reduce the time for an individual transaction by reducing the number of CPU cycles required to process the operations. You declare a T-SQL module to be natively compiled when you create it. As of February 2018, the types of T-SQL modules that can be natively compiled include stored procedures, triggers, and scalar user-defined functions.

Because these objects behave very similarly to their traditional counterparts, you can often use In-Memory OLTP to improve performance while making only minimal changes to the database and the app. You can also have both memory-optimized and traditional disk-based tables in the same database, and run queries across the two.



The following T-SQL script shows an example for each of the above types of objects:

```
-- configure recommended DB option
ALTER DATABASE CURRENT SET MEMORY_
OPTIMIZED _ ELEVATE _ TO _ SNAPSHOT=ON
GO
-- memory-optimized table
CREATE TABLE dbo.table1
( c1 INT IDENTITY PRIMARY KEY
NONCLUSTERED,
    c2 NVARCHAR(MAX))
WITH (MEMORY _ OPTIMIZED=ON)
GO
-- non-durable table
CREATE TABLE dbo.temp _ table1
( c1 INT IDENTITY PRIMARY KEY
NONCLUSTERED,
    c2 NVARCHAR(MAX))
WITH (MEMORY _ OPTIMIZED=ON,
      DURABILITY=SCHEMA _ ONLY)
GO
-- memory-optimized table type
CREATE TYPE dbo.tt _ table1 AS TABLE
( c1 INT IDENTITY,
    c2 NVARCHAR(MAX),
    is _ transient BIT
NOT NULL DEFAULT (0),
    INDEX ix _ c1 HASH (c1) WITH
(BUCKET _ COUNT=1024))
WITH (MEMORY _ OPTIMIZED=ON)
GO
-- natively compiled stored procedure
CREATE PROCEDURE
dbo.usp _ ingest _ table1
    @table1 dbo.tt _ table1 READONLY
    WITH NATIVE _ COMPILATION,
SCHEMABINDING
    AS
    BEGIN ATOMIC
        WITH (TRANSACTION ISOLATION
LEVEL=SNAPSHOT,
            LANGUAGE=N'us _ english')
```

```
DECLARE @i INT = 1

WHILE @i > 0
BEGIN
    INSERT dbo.table1
    SELECT c2
    FROM @table1
    WHERE c1 = @i AND is _ transient=0

    IF @@ROWCOUNT > 0
        SET @i += 1
    ELSE
        BEGIN
            INSERT dbo.temp _ table1
            SELECT c2
            FROM @table1
            WHERE c1 = @i
            AND is _ transient=1

            IF @@ROWCOUNT > 0
                SET @i += 1
            ELSE
                SET @i = 0
        END
END
END

-- sample execution of the proc
DECLARE @table1 dbo.tt _ table1
INSERT @table1 (c2, is _ transient)
VALUES (N'sample durable', 0)
INSERT @table1 (c2, is _ transient)
VALUES (N'sample non-durable', 1)
EXECUTE dbo.usp _ ingest _ table1
@table1=@table1
SELECT c1, c2 from dbo.table1
SELECT c1, c2 from dbo.temp _ table1
GO
```

This script is from the [overview and usage scenarios documentation](#), which covers the In-Memory OLTP technology in both SQL Server and Azure SQL Database.

For additional information, see:

- An introductory blog post on [In-Memory OLTP in Azure SQL Database](#).
- [Documentation, videos, and demos to help you get started](#) with In-Memory OLTP.
- How you can [leverage In-Memory OLTP when using temporary tables, table variables, or table-valued parameters](#).
- A case study with an [in-depth look at a temporary table scenario](#).

Columnstore indexes

Data is everywhere, in every form, and it's growing every day. New data sources are coming online at an ever-increasing rate, leading to new opportunities to gain valuable insights. However, running queries that process large datasets can be slow, especially when using traditional, row-based data structures. Sure, you can create the usual indexes for the types of queries required by canned and ad hoc reports, but what about when that's not enough?

To understand your options, it's worth looking at how data is usually stored in a relational database: in rows. Thus the term *rowstore*, which refers to data that's logically organized as a table with rows and columns, and then physically stored in a row-wise data format. When you create a traditional index in a table, you're creating a *rowstore index*. A rowstore index works best on queries that seek into the data, searching for a particular value, or for queries on a small range of values. Rowstore indexes work well for transactional workloads, which tend to require mostly index seeks instead of full-table scans. However, they're not always ideal for queries that must scan large tables with a lot of data. But what else is there?

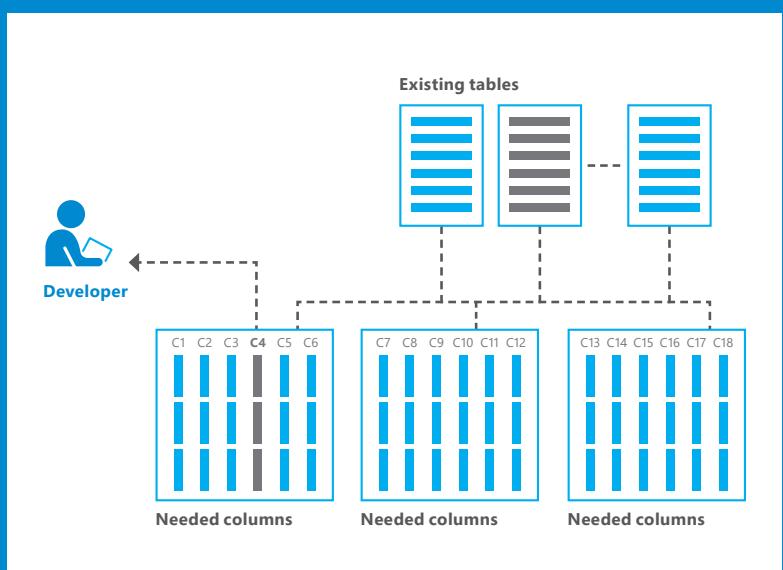
Columnstore indexes, a feature in Azure SQL Database, provide the ability to store, retrieve, and manage data by using an index that's based on a columnar data format, or *columnstore*. Compared to rowstore indexes, when used for queries that need to scan large amounts of data, especially on large tables, a columnstore index can boost query speeds by up to 100 times. And a columnstore index can also provide a much higher level of data compression, typically 10 times what you could

How it works

So just how do columnstore indexes deliver such gains? Columns in a table store data from the same domain and commonly have similar values, which is why applying columnar compression results in high compression rates. High compression rates, in turn, improve query performance by minimizing or eliminating I/O bottlenecks. High compression rates also improve query speeds by using a smaller in-memory footprint, enabling the database to perform more in-memory query and data operations.

Columnstore indexes also reduce I/O and CPU usage in other ways. Remember that queries often select only a few columns from a table. With storage organized by column, only those columns needed by the query are loaded into memory and processed. This is particularly beneficial for wide tables, such as fact tables found in data warehouses and data marts.

In addition, columnstore indexes employ batch-mode processing: queries are processed in batches instead of row by row, and single CPU instructions can manipulate multiple rows at a time. This increases performance by further reducing CPU usage.





otherwise achieve, enabling you to reduce storage costs while improving performance.

Azure SQL Database supports two types of columnstore indexes:

Clustered columnstore indexes, which you can use to reduce your storage footprint and improve performance for data warehouse and historical data scenarios.

Nonclustered columnstore indexes, which you can use to implement real-time operational analytics against a transactional database—that is, a HTAP workload.

It's also worth noting that, in some cases, you can replace several traditional, row-oriented indexes that were used to serve analytical queries with a single columnstore index. This not only improves the performance of analytical queries, but also reduces storage footprint. It can even improve transaction-processing performance because there are fewer indexes to maintain.

Columnstore indexes can also be combined with memory-optimized tables (that is, In-Memory OLTP), enabling you to achieve fast transaction processing and fast analytics queries with the same dataset.

Columnstore indexes are available in all Azure SQL Database databases in the premium tier.

Putting columnstore indexes to use

Creating a columnstore index is easy. The following code sample shows how you can create a nonclustered columnstore index on an existing OLTP table:

```
--This example creates a nonclustered columnstore index on an existing OLTP
table.
--Create the table
CREATE TABLE t_account (
    accountkey int PRIMARY KEY,
    accountdescription nvarchar (50),
    accounttype nvarchar(50),
    unitsold int
);

--Create the columnstore index
CREATE NONCLUSTERED COLUMNSTORE INDEX account_NCCI
ON t_account (accountkey, accountdescription, unitsold)
```

And here's how you can create a memory-optimized table with a columnstore index:

```
-- This example creates a memory-optimized table with a columnstore index.
CREATE TABLE t_account (
    accountkey int NOT NULL PRIMARY KEY NONCLUSTERED,
    Accountdescription nvarchar (50),
    accounttype nvarchar(50),
    unitsold int,
    INDEX t_account_cci CLUSTERED COLUMNSTORE
)
WITH (MEMORY_OPTIMIZED = ON );
GO
```

These code samples are from [getting started with Columnstore for real time operational analytics](#), which discusses how you can run both analytics and OLTP workloads on the same database tables at the same time.

For additional information, see:

- Documentation on [how columnstore indexes work and when to consider using them](#).
- A blog post on the [differences between clustered and nonclustered columnstore indexes](#).
- A list of [other articles on putting columnstore indexes to use](#).

Intelligent performance monitoring and tuning

Achieving top performance isn't just about scalability and resources; it's also about optimization.

Fortunately, Azure SQL Database includes built-in intelligence to automate this for you—by routinely monitoring performance, tuning your database, and optimizing query processing. Automatic performance monitoring and tuning is available across service tiers and performance levels, ready to work on your behalf in the background to make sure you get the most for your money regardless of your database size, structure, or workload.

Intelligent Insights

Monitoring database performance starts with monitoring resource utilization. Sure, you can do this manually by using graphical tools in the Azure portal or using dynamic management views. However, this requires work—and as a manual process, it doesn't scale. Besides, you probably have better things to do than watching a bunch of performance counters and, if one looks off, figuring out what it means and *then* seeing what you can do about it.

With Azure SQL Database, you'll always know what's going on with your database. [Azure SQL Database Intelligent Insights](#) does the work for

you, employing advanced artificial intelligence (AI) to automatically monitor performance and alert you of any issues. Intelligent Insights even identifies the root cause of the issue and, when possible, provides recommendations for resolution. It works whether you have one database or thousands, providing a powerful combination of proactive monitoring and faster mitigation of potential issues to help you optimize database performance, decrease use of DevOps resources, and ultimately reduce your total cost of ownership (TCO).

Putting Intelligent Insights to use

As a smart performance diagnostics log, Intelligent Insights can be easily integrated with other services, including Azure Log Analytics, Azure Event Hubs, Azure Storage, and third-party products. Integration with Log Analytics is one of the easiest ways to get started, enabling you to view the output of Intelligent Insights with a web browser. Integration with Event Hubs is typically used to configure custom monitoring and alerting scenarios. Integration with Storage helps with custom application development, such as custom reporting, data archiving, and data retrieval.

Here's an Intelligent Insights report in Azure SQL Analytics:

The screenshot shows the Microsoft Operations Management Suite interface for Azure SQL Analytics (Preview). The main pane displays a 'Detection' summary with an orange warning icon. It lists two items with 50% impact each, both labeled '1105'. Below this, it provides details about critical impact exceptions detected on 9/17/2017 at 3:15:00 PM, where the database reached its storage size quota. It also mentions errors detected on 9/18/2017 at 12:00:00 PM. The status is 'Active'.

Integration of Intelligent Insights with Azure services is achieved by first enabling Intelligent Insights logging (SQLInsights log), and then configuring Intelligent Insights log data to stream into the desired product or service.

For additional information, see:

- Setting up Intelligent Insights with [Azure Log Analytics, Event Hubs, Azure Storage](#), and [third-party tools or for custom alerting and monitoring](#).
- Using Intelligent Insights to [discover and troubleshoot performance issues](#).
- How Intelligent Insights are generated based on [query duration](#), [timeout requests](#), [excessive wait times](#), or [errored requests](#).

Automatic tuning

Azure SQL Database can do more than just intelligently monitor database performance; it can intelligently optimize your database,

How it works

Intelligent Insights analyzes Azure SQL Database performance by comparing database workload for the past hour with a "baseline" that's calculated based on the past seven days. In doing so, it takes into account those queries determined to be the most significant to database performance, such as the largest and most repeated queries. Because each database is unique in its structure, data, usage, and application, each baseline workload is specific and unique to the individual database instance. Intelligent Insights also monitors absolute operational thresholds and detects issues such as excessive wait times, critical exceptions, and issues with query parameterizations that might affect database performance.

Performance issues are detected via the use of sophisticated AI models, which are based on the behavior of thousands upon thousands of Azure SQL Database instances. The metrics used to measure and detect database performance issues are based on query duration, timeout requests, excessive wait times, and errored-out requests, as explained in the [Detection metrics](#) section of the documentation.

When a performance issue is detected, Intelligent Insights analyzes the root cause and generates a performance diagnostics log with insights into what's happening within your database—including information that can include metadata about the database (such as a resource UDI), observed time range, metrics that caused the insight to be generated, affected queries and error codes, the [detection pattern that triggered the event](#), a root cause analysis in a human-readable format, and, where possible, a recommendation about what to do about it.

Every detected issue is tracked throughout its life cycle, with updates provided in the diagnostic log at 15-minute intervals, making it easy to stay on top of things until performance is back up to par. When a performance issue is detected, the issue is flagged as "Active." After the issue is mitigated, the status is changed to "Verifying." When Azure SQL Database confirms that the issue is resolved, the status is updated to "Complete."

too. With [automatic tuning](#), Azure SQL Database continuously monitors query performance, generates and applies tuning recommendations, and tests each action taken to verify that it contributes to improved performance. If performance doesn't improve, that specific tuning action is undone. The longer a database uses automatic tuning, the better it performs.

As with Intelligent Insights, automatic tuning is AI driven, with Azure SQL Database continually learning horizontally from all the databases it supports. Automatic tuning is safe to use, and it's perhaps one of the most important features you can enable to provide stable, peak-performing workloads. Automatic tuning is designed not to interfere with database workloads. Tuning recommendations are applied only during periods of low utilization and can be temporarily disabled by the system itself to protect workload performance. Of course, you'll have a clear picture of everything that automatic tuning does to improve performance through the tuning recommendations view.

There are two main aspects to automatic tuning in Azure SQL Database:

Automatic index management identifies indexes that should be added or removed. There are two settings:

- **CREATE INDEX**, which identifies the indexes that might improve the performance of your workload, creates the indexes, and verifies that they have improved the performance of the queries.

- **DROP INDEX**, which identifies redundant and duplicate indexes in addition to indexes that weren't used for long periods of time.

Automatic plan correction identifies and addresses problematic SQL query execution plans. When automatic plan correction is enabled via the **FORCE LAST GOOD PLAN** setting, if a query execution plan is identified to be slower than the previous good execution plan, the query execution plan is reverted back to the last known good plan instead of using the regressed query execution plan.

Putting automatic tuning to use

Prior to 2018, automatic tuning could be manually enabled on your database subscription (see [Enable automatic tuning for more information](#)). In 2018, automatic tuning will be on by default, as described in [Automatic tuning will be a new default](#), and this setting will gradually roll out across the entire Azure platform.

When enabled, automatic tuning can function autonomously. If you want more control, you can turn off the automatic application of tuning recommendations and apply them manually through the Azure portal. It's also possible to manually apply automatic-tuning recommendations using the scripts and tools of your choice.

Automatic-tuning options (CREATE INDEX, DROP INDEX, and FORCE LAST GOOD PLAN) can be independently turned on or off per database, or they can be configured on a logical server and applied on every database that

inherits settings from the server—making it easy to manage automatic-tuning options across many databases. You can [enable automatic tuning at the server level using the Azure portal](#) or the [REST API](#). You can also [enable automatic tuning at the database level using the Azure portal](#), the [REST API](#), or [T-SQL](#) (equivalent to configuring it through the portal).

For example, to enable automatic tuning on a single database via T-SQL, connect to the database and execute the following:

```
ALTER DATABASE current SET  
AUTOMATIC_TUNING = AUTO  
/* possible values AUTO,  
INHERIT and CUSTOM */
```

In this example, setting automatic tuning to AUTO will apply Azure defaults. By setting it to INHERIT, the configuration of automatic tuning will be inherited from the parent server. If you choose CUSTOM, you'll need to manually configure automatic tuning, which you can do using the following T-SQL code:

```
ALTER DATABASE current  
SET AUTOMATIC_TUNING (   
FORCE_LAST_GOOD_PLAN =  
[ON | OFF | DEFAULT],  
CREATE_INDEX = [ON | OFF |  
DEFAULT],  
DROP_INDEX = [ON | OFF |  
DEFAULT])
```

In this example, setting any individual tuning option to ON will override the setting that the database inherited and enable the tuning option. Similarly, setting it to OFF will override the inherited setting and disable the tuning option. Setting it to DEFAULT will use the inherited setting.

For additional information, see:

- [How automatic tuning works](#) in Azure SQL Database and SQL Server 2017.
- How to [enable automatic tuning](#) at the server or database level.
- How to [configure automatic tuning using T-SQL](#).
- How to [manually review and apply automatic-tuning recommendations](#).
- A blog article on [how various organizations have used automatic tuning](#).

Adaptive query processing

The built-in intelligence in Azure SQL Database goes further than monitoring or tuning. Azure SQL Database is intelligent enough to adapt its query processing to your specific database workload, at a level you might not even be aware of. Ever heard of batch mode memory grant feedback, batch mode adaptive joins, or interleaved execution? If so, you'll appreciate what Azure SQL Database does for you automatically in these areas. If not, don't worry—Azure SQL Database has still got your back.

Batch mode memory grant feedback, batch mode adaptive joins, and interleaved execution are all examples of the [adaptive query processing](#) family of features in Azure SQL Database—the first three to be introduced, in fact. They're designed to improve query processing by applying "learn and adapt" query optimization strategies to your database's runtime workload, helping to address performance issues related to historically intractable query optimization problems.

At a high level, before Azure SQL Database executes a specific query, the query optimization process generates a set of feasible execution plans. The “cost” of each option is estimated, and the plan with the lowest estimated cost is used for query execution. But sometimes the plan chosen by the query optimizer isn’t optimal; for example, the total number of rows processed at each level of the query plan (referred to as the cardinality of the plan) might be incorrect, leading to an incorrect estimation of the query-processing cost for that plan. Because estimated processing costs help determine which plan is selected, incorrect cardinality estimates might result in the selection of a nonoptimal plan.

Putting adaptive query processing to use

You can make your database workloads automatically eligible for adaptive query processing by setting your database compatibility level to 140, as illustrated in the following T-SQL example:

```
ALTER DATABASE
[WideWorldImportersDW] SET
COMPATIBILITY _ LEVEL = 140;
```

This example is from the [documentation on adaptive query processing](#), which covers each of the three cases in greater detail. You can also find more information in the blog articles on [batch mode adaptive joins](#), [batch mode memory grant feedback](#), and [interleaved execution for multi-statement table valued functions](#).

How it works

Let’s take a closer look at how adaptive query processing works in each of the three cases:

Batch mode memory grant feedback.

A query’s post-execution plan includes the minimum required memory needed for execution and the ideal memory grant size to have all rows fit in memory. However, performance suffers when memory grants are incorrectly sized; excessive grants waste memory and thus reduce concurrency, whereas insufficient memory grants cause expensive spills to disk. By addressing repeating workloads, batch mode memory grant feedback recalculates the actual memory required for a query and then updates the grant value for the cached plan. When an identical query statement is executed, the query uses the revised memory grant size instead of the original one.

Batch mode adaptive joins.

This adaptive query-processing feature enables the choice of a hash join method or nested loop join method to be deferred until after the first input has been scanned. The adaptive join operator defines a threshold that’s used to decide when to switch to a nested loop plan, enabling the plan to dynamically switch to a better join strategy during execution. If the row count of the build join input is small enough that a nested loop join would be more optimal than a hash join, the plan switches to a nested loop algorithm. If the build join input exceeds a specific row count threshold, no switch occurs, and the plan continues with a hash join.

Interleaved execution for multi-statement table valued functions.

Interleaved execution changes the unidirectional boundary between the optimization and execution phases for a single-query execution and enables plans to adapt based on the revised cardinality estimates. During optimization, if a candidate for interleaved execution is encountered (currently these are multi-statement table valued functions, or MSTVFs), Azure SQL Database will pause optimization, execute the applicable subtree, capture accurate cardinality estimates, and then resume optimization for downstream operations.

Conclusion

No matter what your app does, you need to ensure that it performs. When you develop with Azure SQL Database, you won't need to spend a lot of time on database performance. You can dynamically scale database resources at any time, knowing that performance is already optimized to help you get the most out of those resources you're paying for. And as your app runs, Azure SQL Database automatically monitors and tunes your database for you, employing sophisticated AI to continually optimize performance and adapt to changing workloads.

With Azure SQL Database, the intelligent database for developers, get back to what you love: coding.

Get \$200 to try Azure SQL Database with an [Azure free account](#), and then [watch the video](#) to create your first database in just a few minutes.