Microsoft SQL Server

Microsoft

Mission-Critical Performance

Technical White Paper

Published: March 2016

Applies to: Microsoft SQL Server 2016

Summary: As the volume and complexity of data continue to increase, organizations require a new approach to their mission-critical capabilities. This white paper examines how capabilities built into Microsoft SQL Server—including enterprise-grade performance, security, availability, and scalability—define a "new mission critical" that answers what kinds of capabilities organizations need to compete in a dynamic global landscape.

Copyright

Contents

Microsoft SQL Server evolution

By developing and enhancing new features, Microsoft SQL Server continues to evolve and stay ahead of organizational data needs. Customers have responded to this evolution by showing confidence in using SQL Server to manage their mission-critical data. Industry analysts have also responded positively. For example, Gartner recently rated SQL Server as having the most complete vision of any operational database management system (Figure 1).

Figure 1: Gartner Magic Quadrant for operational database management systems (2015)

In addition, SQL Server has consistently added groundbreaking functionality over the last 15 years (Figure 2).

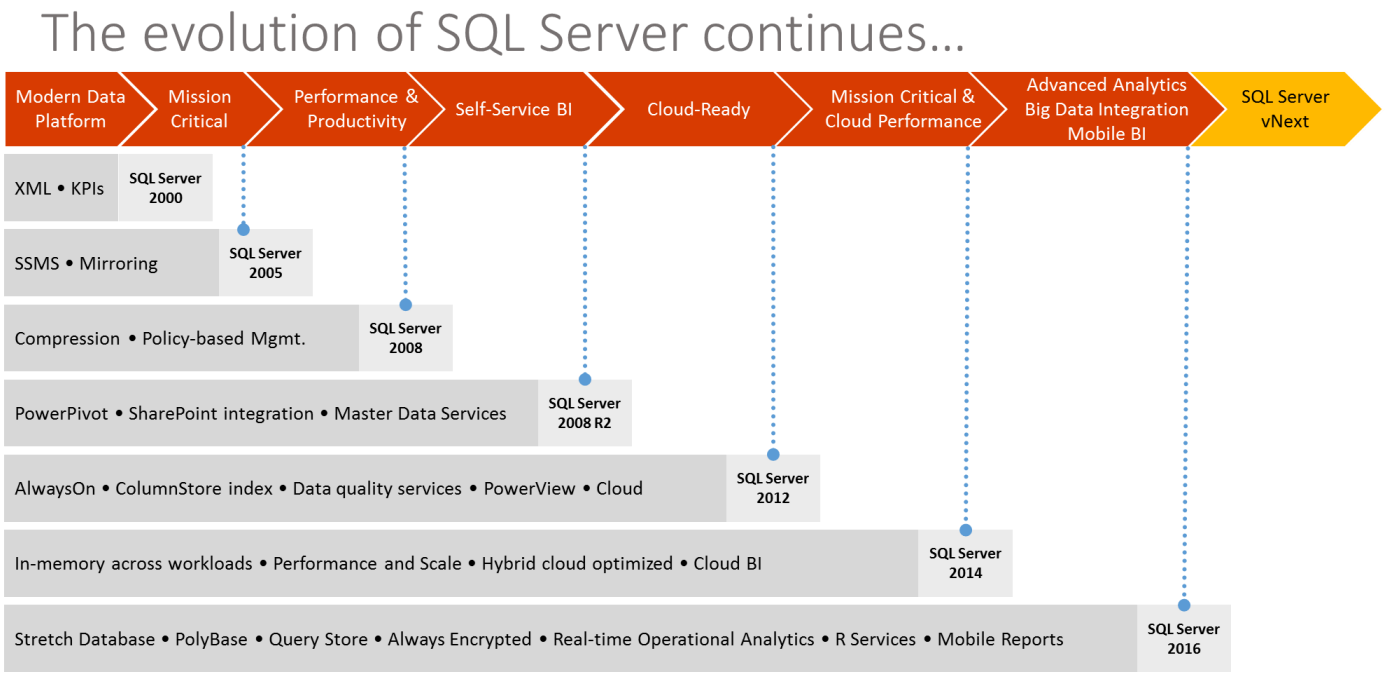


Figure 2: SQL Server functionalities added across releases

SQL Server 2016 introduces many new features and enhancements, including:

- Performance improvements via In-Memory OLTP (online transaction processing), Real-Time Analytics, Query Store, and more
- Security improvements via Always Encrypted and Row-Level Security
- Availability improvements via failover clustering, availability groups, and AlwaysOn improvements
- Scalability, security, and availability improvements through support for Windows Server and Server Core

Mission-critical performance with SQL Server

SQL Server's evolution mirrors the growing critical nature of data for enterprises. Data is the new currency, and it has become a major competitive differentiator. Companies with a data-centric culture are harnessing increasingly diverse data. This data is not just relational or internal but includes both relational and non-relational, as well as internal and external data sources. These companies are using new analytical models to review historical data and apply it to help predict the future. Insights from the data are many, which are shared more broadly in the organization. And, in many cases, much of this analysis is dispensed at near real-time speeds. Companies with a data-centric culture are more productive, more efficient with their operations, and they innovate faster—all leading to improvements in their bottom line.

More than ever, organizations need mission-critical operations that are easy to deploy and that are balanced with faster time-to-solution. This white paper reviews four main areas in which SQL Server continues to deliver in this manner: performance, security, availability, and scalability.

Performance

SQL Server consistently leads in performance benchmarks, such as TPC-E and TPC-H, and in real-world application performance. With SQL Server 2016, performance is enhanced with a number of new technologies, including in-memory enhancements, Query Store, and temporal support, to name a few.

SQL Server's integrated in-memory toolset goes far beyond isolated features to support dramatic performance improvements in a wide range of scenarios. These technologies include In-Memory Online Transaction Processing (In-Memory OLTP), primarily for transactional workloads, In-Memory Columnstore for decision-support workloads (discussed in the Microsoft SQL Server: Deeper Insights Across Data white paper), and the new Query Store feature, which allows you to monitor and optimize query plans for particular application scenarios over time—providing additional performance-tuning opportunities.

In-Memory Online Transaction Processing

In-memory technology for SQL Server dramatically improves the throughput and latency of SQL Server OLTP capabilities. It is designed to meet the requirements of the most demanding transaction processing applications, and Microsoft has worked closely with a number of companies to prove these gains. The feature set of In-Memory OLTP includes the following:

- **Memory-optimized tables**: There are two types of memory-optimized tables. Durable tables are fully logged and persist over server restarts. Non-durable tables do not persist over server restarts and are most commonly used in lieu of global temp tables in the user database or in scenarios where persistence is not needed, such as staging tables in an Extract Transform Load (ETL) process.
- **Memory-optimized table variables**: These variables are created using memory-optimized table types. Variables are stored in-memory, leading to more efficient data access because they use the

same memory-optimized algorithms and data structures as memory-optimized tables—particularly when using natively compiled stored procedures.

- **Natively compiled stored procedures**: SQL Server can natively compile stored procedures that access memory-optimized tables. Native compilation enables faster data access and more efficient query execution than interpreted (traditional) Transact-SQL. Natively compiled stored procedures are parsed and compiled when they are loaded to native DLLs (dynamic-link libraries). This is in contrast to other stored procedures that are compiled on first run. They have an execution plan created and reused, and they use an interpreter for execution.
- **Natively compiled scalar user-defined functions (UDFs)**: These replace traditional scalar UDFs that do not perform data access, and this replacement reduces UDF runtime. Natively compiled scalar UDFs cannot access disk-based tables. If data access is required, consider migrating the table to memory-optimized (if no data access occurs, migration is not required).

In-Memory OLTP is designed on the following architectural principles:

- **Optimize for main-memory data access**. Storage-optimized engines (such as the current OLTP engine in SQL Server) will retain hot data in a main-memory buffer pool based on frequency of access. The data access and modification capabilities, however, are designed so that data may be paged in or out to disk at any point. With In-Memory OLTP, you place tables used in the extreme transaction-processing portion of an application into memory-optimized main-memory structures. The remaining application tables, such as reference data details or historical data, are left in traditional storage-optimized structures. This approach enables you to optimize hotspots for memory use, without having to manage multiple data engines. Main-memory structures for In-Memory OLTP eliminate the overhead and indirection of the storage-optimized view while still providing the full atomicity, consistency, isolation, and durability (ACID) properties you expect from a database system.
- **Include tools for migration**. To identify the appropriate tables and memory structures for utilizing In-Memory OLTP, SQL Server 2016 Management Studio includes tools designed to assist users in transitioning to In-Memory OLTP. These include transaction performance analysis to identify objects that would benefit from migration, migration advisors to assist in migrating disk-based tables to memory-optimized tables, and migration of traditional stored procedures and functions to natively compiled objects.
- **Accelerate business-logic processing**. In-Memory OLTP, queries, and procedural logic in procedures that are stored in Transact-SQL (T-SQL) are compiled directly into machine code through aggressive optimizations that are applied at compilation time. Consequently, the stored procedure can be executed at the speed of native code.
- **Provide frictionless scale-up**. In-Memory OLTP implements a highly scalable concurrency control mechanism and uses a series of lock-free data structures to eliminate traditional locks and latches while guaranteeing the correct transactional semantics that ensure data consistency.
- **Integrate into SQL Server**. One of the most impressive things about In-Memory OLTP is that it achieves breakthrough improvements in transactional processing capabilities without requiring a separate data management product or new programming model. This enables an integrated developer and database administrator (DBA) experience with the same T-SQL, client stack, tooling, backup and restore, and AlwaysOn capabilities. By offering in-memory functionality

within SQL Server, your total cost of ownership ends up being lower than it would be if you were to purchase, manage, and maintain a separate system for handling in-memory processing.

In-Memory OLTP enhancements in SQL Server 2016

Performance and scaling improvements

Improvements to In-Memory OLTP enable scaling to larger databases and higher throughput in order to support bigger workloads. In addition, a number of limitations on tables and stored procedures have been removed to make it easier to migrate your applications and leverage the benefits of In-Memory OLTP. Scalability improvements include:

- Multiple threads to persist memory-optimized tables
- Multi-threaded recovery and merge operations
- Dynamic management views
- Parallel plan support for accessing memory-optimized tables using interpreted T-SQL
- Parallel support for hash indexes

Transact-SQL improvements

Query surface area in native modules has been improved to include support for subqueries in SELECT statements, nested execution of natively compiled modules, natively compiled inline table-valued functions, built-in security functions, and increased support for built-in math functions. Additional native support for query constructs such as UNION, DISTINCT, OUTER JOIN, OR, and NOT is now available. Memory-optimized tables now support nullable index key columns; large object (LOB) data types; constraints such as FOREIGN KEY, CHECK, and UNIQUE; and triggers (AFTER) for insert, update, and delete operations.

For more information: Supported features

Cross-feature support

System-versioned temporal tables provide a solution for scenarios such as data auditing and point-in-time analysis of OLTP workloads. System-versioned temporal tables for memory-optimized tables are able to provide high transactional throughput and lock-free concurrency while simultaneously storing large amounts of historical data. Implementing system-versioned temporal tables with memory-optimized tables is simple and provides an easier route than manual implementation for maintaining historical data.

For more information: System-versioned temporal tables

Query Store allows you to monitor the performance of natively compiled code for workloads running In-Memory OLTP. Compile and runtime statistics are collected and exposed just as they are for disk-based workloads. You can continue using Query Store views in SQL Server Management Studio as well as custom scripts you have developed for disk-based workloads on migrated In-Memory OLTP workloads. This saves your investment in learning Query Store technology and makes it generally usable for troubleshooting all types of workloads.

For more information: Query Store with In-Memory OLTP

Row-Level Security (RLS) is supported for memory-optimized tables. RLS is a new feature introduced in SQL Server 2016 that allows developers to restrict access to rows based on filter and block predicates. Filter predicates seamlessly exclude rows from query results, returning only those records for which a user is authorized. Block predicates prevent unauthorized users from making changes to records for which a user is not authorized, particularly in the case of bulk update/delete commands. The execution of both predicate types is transparent to end users.

For more information: Row-Level Security in memory-optimized tables

Multiple Active Result Sets (MARS) support using queries and natively compiled stored procedures. MARS enables data requests from multiple queries without the need to retrieve each result set before sending a request to fetch rows (via MARS-enabled connection).

For more information: MARS

Transparent Data Encryption (TDE) improvements include support for the storage of memory-optimized tables that can be encrypted to enable TDE of the database.

In-memory analytics

With the explosion of devices (such as mobile phones, smart meters, and smart cars) generating data, organizations are faced with the challenge of managing exponentially growing data while still delivering effective, interactive data-analysis response times for making faster business decisions. The traditional storage model for storing data continues to work well for transactional workloads, but it increasingly falls short when processing large numbers of rows efficiently for analytics workloads.

SQL Server 2012 addresses these challenges with an alternate storage model that stores data as columns rather than rows. This means you can leverage columnstore index to achieve high data compression, typically 10 times more, while speeding up the performance of analytics queries by up to 100 times.

Figure 3 shows a simple relational table stored as individual columns. Each white box represents approximately 1 million values of a column stored together. The values in a column across multiple rows are drawn from the same domain. More compression can be achieved in this manner than with page compression, for example, where data is stored as rows.

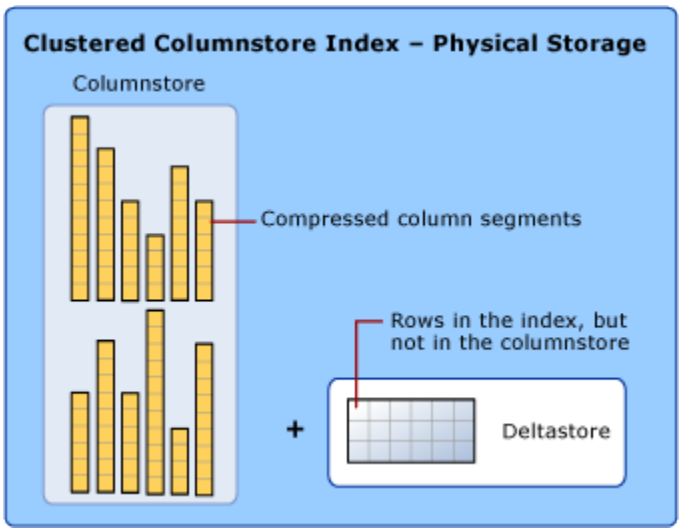


Figure 3: Conceptual diagram of a columnstore

In addition to reducing storage, this can speed up analytic queries (which typically scan millions or even billions of rows) by reducing I/O significantly. Columnstore storage also enables the fetching of only those columns needed to execute the query. This, in turn, can further reduce the I/O. For example, if the analytics query only accesses columns C1, C2, and C3, the other columns—C4 and C5—are skipped. This differs from a rowstore, where all columns in a row must be fetched regardless of the references in the query. Speed can be increased even more (typically four times) with a new batch mode execution mode that processes a set of rows (typically around 1,000) together. Perhaps the biggest benefit is that no application changes are required once you migrate your application to use columnstore index. All queries run unchanged but significantly faster.

The columnstore index in SQL Server 2016 has been significantly enhanced with the following key improvements:

- Ability to create one or more traditional nonclustered indexes on a table with a clustered columnstore index. This enables users to enforce primary and foreign key (PK/FK) constraints and speed up the query performance for short-range queries.
- Ability to use RCSI (read-committed snapshot) and SI (snapshot isolation) to get transactional consistent results with no blocking with concurrent DML (data manipulation language) operations.
- Ability to offload analytics workload to readable secondary in AlwaysOn configuration.
- Improved analytics query performance with string predicate pushdown, aggregate pushdown, and new batch mode operators such as SORT and Window Aggregates. You will get these benefits with no changes to your analytics application after you upgrade to SQL Server 2016.
- Online columnstore index maintenance to defragment the index by removing the deleted rows.
- Improved monitoring and troubleshooting with new DMVs (dynamic management views), extended events (XEvents), and performance counters.
- Updateable nonclustered columnstore index (NCCI) to enable real-time operational analytics (described in detail in the next section).

Real-time operational analytics

SQL Server 2016 introduces real-time operational analytics—the ability to run both analytics and transactional workloads on the same database tables concurrently. SQL Server enables you to create an updateable nonclustered columnstore index on traditional rowstore tables. Transactional workloads run against the rowstore, while analytics workloads run against the columnstore index (Figure 4). SQL Server automatically maintains all changes to the indexes, so the transactional changes are always up-to-date for analytics. The time it takes to maintain a columnstore index can be minimized or eliminated by using compression delay or by creating a filtered columnstore index on colder data. These factors make it possible and practical to run analytics in real-time as opposed to traditional data warehouse implementations that require data migration/transformation from a transactional system to a dedicated data warehouse. Real-time operational analytics is supported both on disk-based and memory-optimized tables.

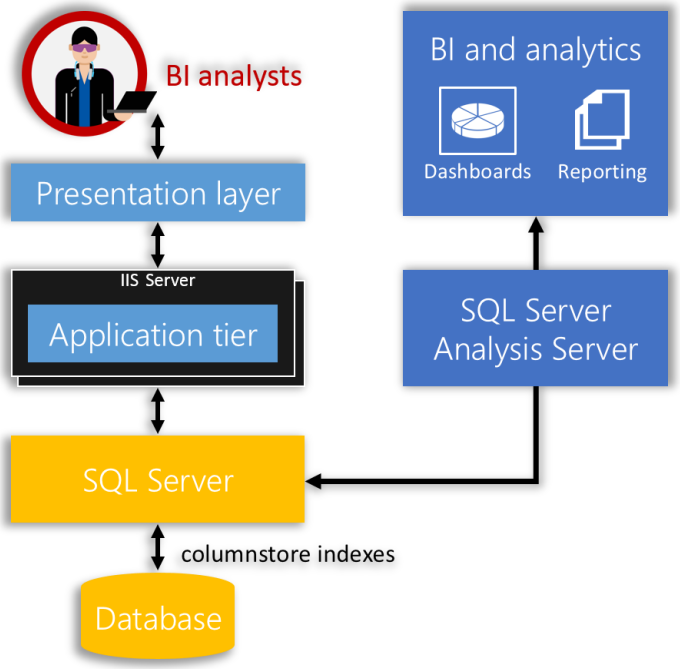


Figure 4: Running of analytics on columnstore indexes

Real-time operational analytics for disk-based tables

To get started with real-time analytics, you simply create a nonclustered columnstore index for analytics queries and drop all other indexes that were created for analytics. You do not need to change your application, because the query optimizer estimates the performance cost and will automatically choose the best index for each query. Analytics queries will run against the columnstore index, and the transactional workload continues to run against the rowstore using B-tree indexes (Figure 5). The transactional workload runs without requiring any changes but may incur additional overhead to maintain the columnstore.

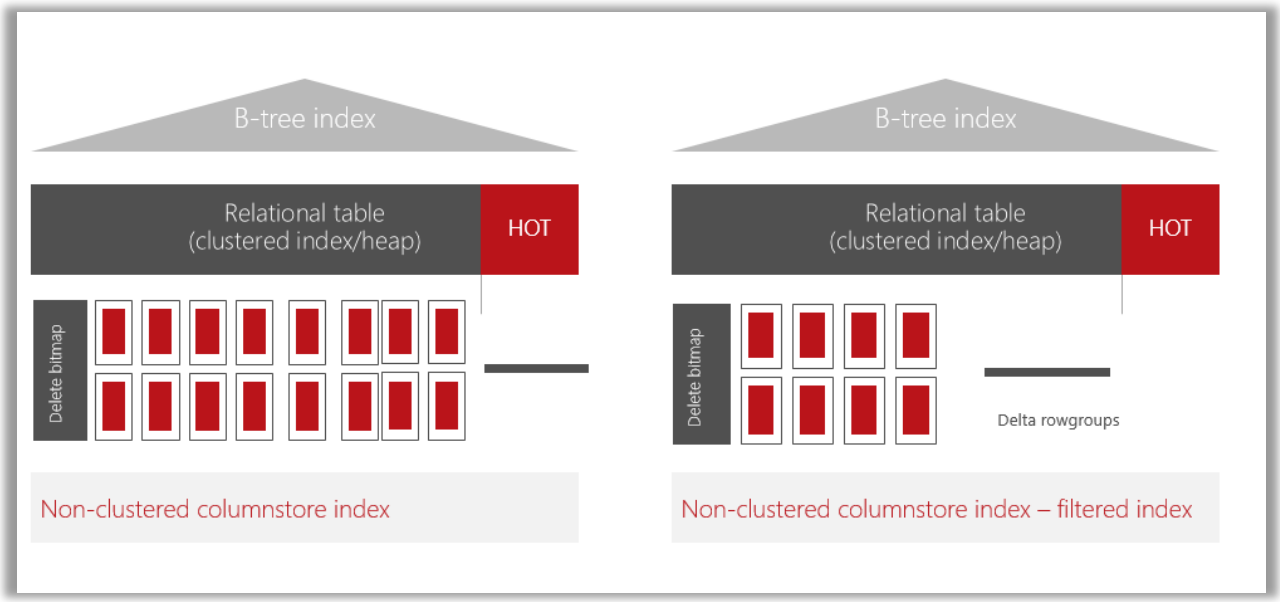


Figure 5: Operational analytics with nonclustered columnstore index (regular and filtered)

A filtered index can be used for minimizing columnstore overhead. You can create a columnstore index on cold data slowly, using a filtered predicate to minimize maintenance. Analytics queries access both columnstore and "hot" data transparently. For example, in an order management application, you can choose to create a columnstore index on orders that have already shipped. Once an order has shipped, it is changed infrequently and can be considered "warm" data. This minimizes the impact of maintaining the columnstore index for the highly active, hot OLTP operations.

Additionally, if you are running your transactional workload in the AlwaysOn configuration, you can offload the analytics to a readable secondary replica to minimize impact on the transactional workload running on the primary replica.

Real-time operational analytics with in-memory tables

SQL Server 2016 allows columnstore index on memory-optimized tables. The columnstore index on an in-memory table enables real-time operational analytics by integrating In-Memory OLTP and In-Memory columnstore technologies to deliver high performance for transactional and analytics workloads. You can create a columnstore index on a memory-optimized table, but all columns must be included in the columnstore. Unlike a nonclustered columnstore index for disk-based tables, there is no explicit delta rowgroup. The data rows that have not yet transitioned into columnstore index are accessed directly from memory-optimized tables and are referred to as "tail" rows (Figure 6). No columnstore index overhead is incurred if the transactional workload accesses rows in the tail. SQL Server allows you to control how long the rows stay in the tail. For example, if the rows stay hot for your workload (that is, get updated multiple

times) for two hours, you can set compression delay to two hours. A background task can then migrate rows from tail to columnstore in chunks of 1 million rows.

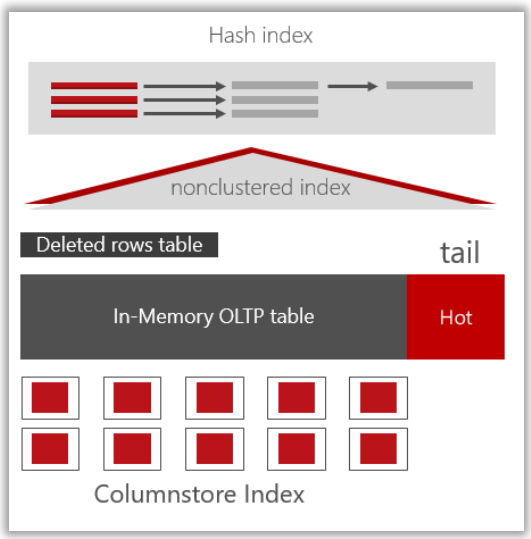


Figure 6: Columnstore on in-memory tables

Key enhancements have been introduced to columnstore indexes that enhance data warehousing scenarios and enable real-time analytics. Memory-optimized tables now support one columnstore index—previously only available to disk-based tables. Nonclustered columnstore indexes (NCCI) support a filtered condition, which enables creation of the NCCI on "cold" data for an operational workload—minimizing the performance impact of having a columnstore index on an OLTP workload. Clustered columnstore indexes now support both primary and foreign keys.

For more information: Columnstore indexes

Query-processing enhancements

Better query results and significantly faster data loading are among the recent improvements to SQL Server.

Cardinality estimator

The cardinality estimator, introduced in SQL Server 2014, improves the querying process and offers the following benefits:

- **Consistent, predictable query performance**. Different operator trees that represent the same relation have the same cardinality estimates.
- **New model for better performance**. Significant changes to the model result in more accurate cardinality estimates and better plan choices.
- **Easier to support**. The querying process separates querying into two steps: decision making and execution. It also produces tracing output to facilitate troubleshooting.

Improvements that trigger automatic updates of statistics enable better query results. Statistics are refreshed more quickly and frequently as a result of the invalidation threshold, which has been set to 20 percent of a single partition. In addition, data loading is significantly faster, because data insertion into a table occurs in parallel through the SELECT INTO operation.

Reduced database size and increased performance: data and backup compression

Many organizations want to increase speed and reliability by putting more data onto specialized disk arrays or a storage area network (SAN), but they are often prohibited by the cost of these high-end disk resources. Backup and data compression in SQL Server can free up space by dramatically reducing the size of databases. Reduced data size can also increase performance. With additional space, more data can be stored on the SAN. Since storing data on the SAN is more reliable, it also increases availability.

SQL Server also enables data compression for people who use Unicode UCS-2. This capability enables organizations that have global language sets in their data storage to make use of data compression and experience the benefits of compression.

Additionally, SQL Server supports columnstore indexes to greatly improve data compression as well as query performance. With SQL Server 2016, columnstore indexes enable real-time analytics on a transactional workload.

For more information: Columnstore indexes

Proactive troubleshooting and diagnostics: Performance Data Collector and Management Studio

To ensure the best possible performance, organizations need to proactively manage the health of their systems and the quality of queries across their digital environments. SQL Server delivers a suite of diagnostics and tuning tools built in at no extra cost. Performance Data Collector allows administrators to view SQL Server diagnostics from performance counters, dynamic management views (DMVs), SQL Trace, and other sources for baseline and historical comparisons. They can view performance data with built-in reports on topics such as server activity, disk usage, and query activity. Additionally, SQL Server Profiler can capture server events for real-time diagnosis, and it can correlate traces with performance counters to analyze events and diagnose issues. Dynamic management views and functions that relay server state information help IT administrators monitor the health of server instances, diagnose problems, and tune performance. The Database Engine Tuning Advisor helps administrators select and create an optimal set of indexes, indexed views, and partitions without requiring an expert understanding of database structure or the internal workings of SQL Server. Users simply select databases to tune, and the advisor generates indexing and partitioning recommendations.

New in SQL Server 2016

SQL Server Query Store

Query Store is a new SQL Server component that captures queries, query plans, runtime statistics, and more in a persistent store inside the database. It is a database-scoped persistent store of query workload history. You can think of it as a flight recorder, or black box, for your database. It can also enforce policies to direct the SQL Server Query Processor to compile queries to be executed in a specific manner, such as forcing plans.

Query Store primarily targets administrative scenarios for performance troubleshooting and identifying regressed workloads. It also collects query texts and all relevant properties, as well as query plan choices and performance metrics. This collection process works across restarts or upgrades of the server and across recompilation of indexes, providing many configurable options for customization. Query Store integrates with existing query execution statistics, plan forcing, and manageability tools. It is a dedicated store for query workload performance data and captures the history of plans for each query. It also captures the performance of each plan over time and persists the data to disk (works across restarts, upgrades, and recompiles). Query Store enables you to:

- Quickly find and fix a plan performance regression by forcing the previous query plan. It can fix queries that have recently regressed in performance due to execution plan changes.
- Determine the number of times a query was executed in a given time window and assist a DBA in troubleshooting performance resource problems.
- Identify costly queries (by execution time, memory consumption, and so forth) in the past "x" hours, and audit the history of query plans for a given query.
- Analyze the resource (CPU, I/O, and Memory) usage patterns for a particular database.
- Maintain performance stability during upgrade to SQL Server 2016.
- Perform A/B testing and back up changes with performance data.

The Query Store can be viewed and managed through Management Studio or by using views and procedures. There are seven Query Store Catalog Views that can present information about the Query Store. There are also various procedures to follow when configuring the Query Store:

- **sys.database_query_store_options**: Returns query store options for a given database.
- **sys.query_context_settings**: Contains information about the semantics affecting context settings associated with a query.
- **sys.query_store_plan**: Contains information about each execution plan associated with a query.
- **sys.query_store_query**: Contains information about the query and its associated overall aggregated runtime execution statistics.
- **sys.query_store_query_text**: Contains the Transact-SQL text and the SQL handle of the query.
- **sys.query_store_runtime_stats**: Contains information about the runtime execution statistics information for the query.
- **sys.query_store_runtime_stats_interval**: Contains information about the start and end times of each interval over which runtime execution statistics information for a query has been collected.

For more information: Query Store

By combining these objects in various queries, you can gain necessary insights about user workload for the period of time when Query Store was active. Query Store also has a rich user-interface component that uses several of these same DMVs (Figure 7). Familiarizing yourself with the UI can be an excellent way to become acquainted with the kind of information obtainable by querying the DMVs.

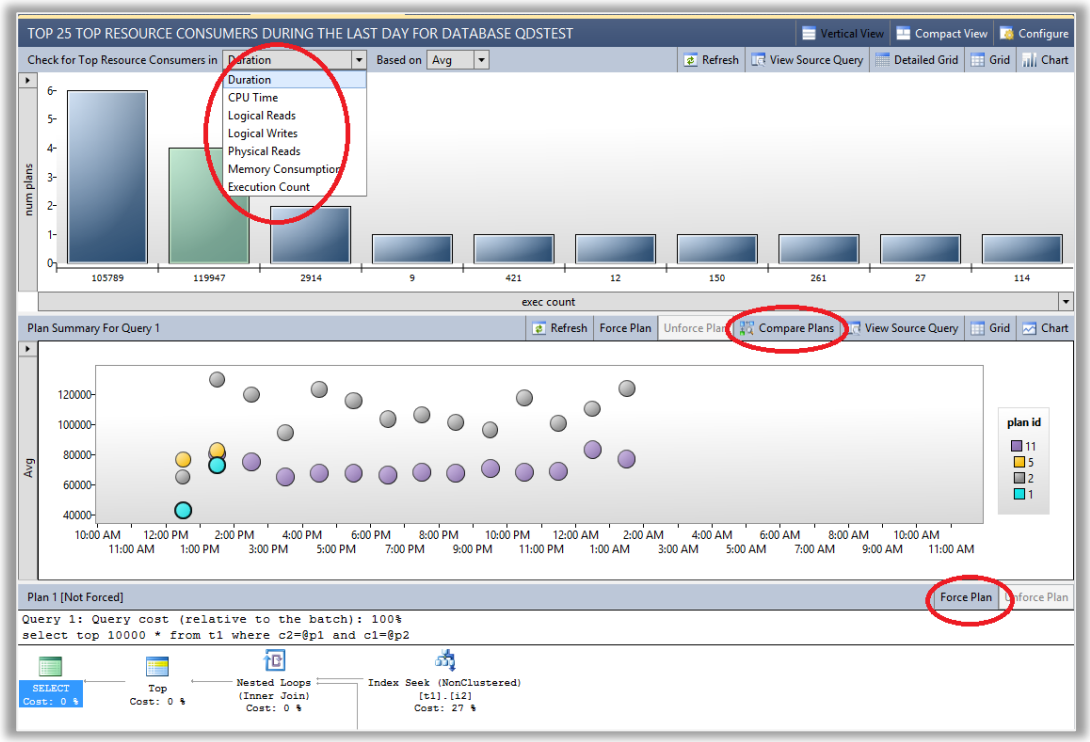


Figure 7: Dynamic management views in Query Store

Live Query Statistics

Before SQL Server 2016, developers and DBAs would have to troubleshoot query performance with Showplan at query runtime. This runtime execution plan, often referred to as query execution statistics, enables you to collect metrics about the query that occurred during its execution (such as its execution time and actual cost) after the query finishes running.

SQL Server 2016 has a new feature—Live Query Statistics (LQS)—that allows you to view what is happening during the query execution (Figure 8). LQS lets you view a list of active queries and associated statistics, such as current CPU/memory usage, execution time, query progress, and so on. This enables rapid identification of potential bottlenecks for troubleshooting query performance issues. LQS also allows users to drill down into a query plan of an active query and view live operator-level statistics, such as the number of generated rows, elapsed time, operator progress, and live warnings. This facilitates in-depth troubleshooting of query performance issues without forcing you to wait for query completion, so you can watch the statistics change during the query execution in real-time.

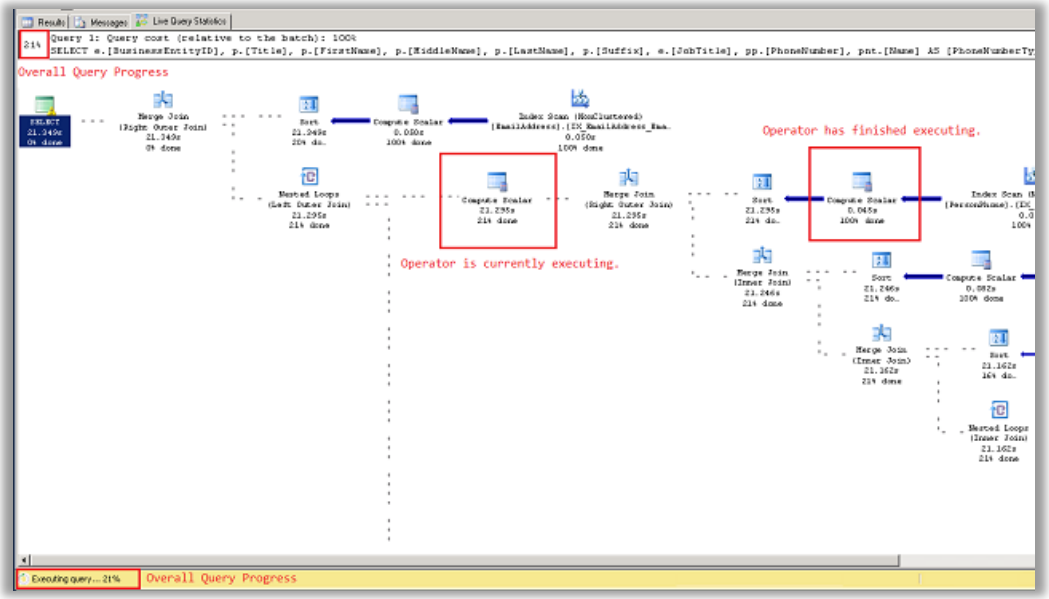


Figure 8: Live Query Statistics in SQL Server 2016

The live execution plan can also be accessed from the Activity Monitor (Figure 9).

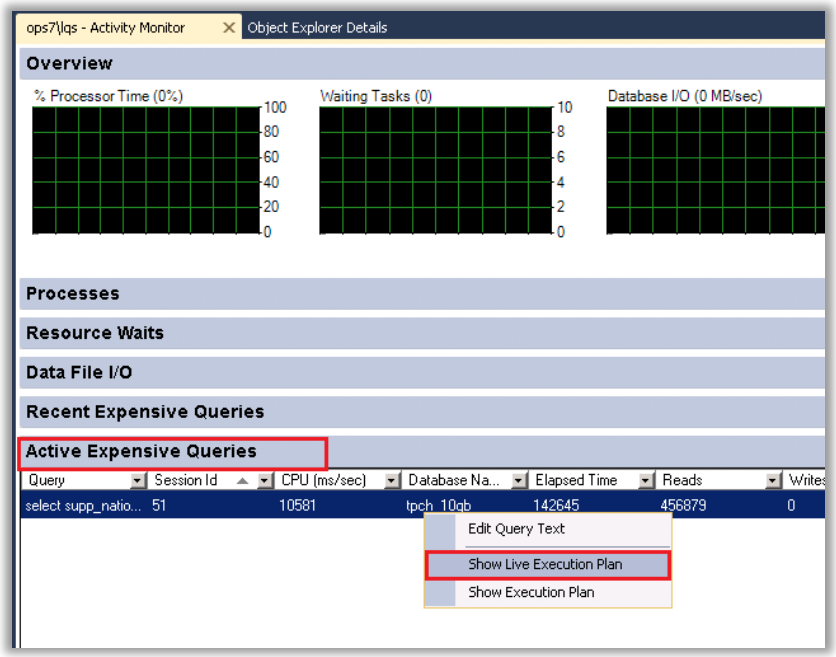


Figure 9: Live execution plan in SQL Server 2016

Temporal database support

Data is rarely static. Seeing how data has evolved over time—or querying data as of a particular point in time—can be very valuable. SQL Server 2016 promises to simplify this process with a new feature called Temporal Tables. With this feature, one can specify that the data history for a table be stored

automatically and transparently by SQL Server and, just as automatically, be retrieved through queries using a new set of T-SQL commands. Furthermore, this can be enabled without requiring changes to any existing applications.

Data audits

Temporal system-versioned tables keep track of data changes, particularly what changes were made, when, and by whom. This level of data tracking lends itself to performing periodic data audits, to gaining insights to how and when specific data changes were introduced. Since existing tables can be configured as temporal system-versioned, auditing capabilities can be introduced to existing data-driven applications with minimal effort.

Point-in-time analysis

Temporal tables can also support scenarios in which users want to see how data sets change over a defined time period. Some examples include describing trends for important indicators past and present, viewing a snapshot of data for a specific point in the past, or determining the differences between two time points for comparison.

Slowly changing dimensions

Temporal tables can be used to reduce the complexity of Slowly Changing Dimensions (SCD). Traditionally, maintaining SCD history required schema overhead in the form of additional columns or a separate table. System-versioned temporal tables eliminate the need for maintaining this overhead.

Recovering data

Recovering lost or corrupted data can be an arduous, time-consuming process. Temporal tables enable pinpoint data recovery, without resorting to backup restores or complex (sometimes error-prone) restore procedures. System-versioned temporal tables offer two additional benefits: operational efficiency, as the database remains online for all workloads, and traceability, as any repair operations themselves are versioned.

For more information: Example scenarios

With system-versioned temporal tables, the history table may increase database size more than regular tables. A large and ever-growing history table can become an issue, both due to pure storage costs, as well as imposing a performance tax on temporal querying. Hence, developing a data retention policy for managing data in the history table is an important aspect of planning and managing the lifecycle of every temporal table.

With SQL Server 2016, the following three approaches—each with its own strengths—are available for managing historical data in the temporal history table:

- Stretch Database
- Table Partitioning
- Custom Cleanup Script

For more information: Data retention management for history table

How does it work?

A temporal table (Figure 10) is also referred to as a system-versioned table. Each temporal table (or "system-versioned" temporal table) consists of two tables: one for current data, and the other for historical data. Within each of these tables, two additional DateTime (DateTime2 datatype) columns are used to define the period of validity for each record: a system start time (SysStartTime) column, and a system end time (SysEndTime) column. The current table contains the current value for each record. The history table contains previous values for each record, if any, and the start and end times for the period for which it was valid.

- **INSERTS**: On an INSERT, the system sets the value for the SysStartTime column to the UTC time (coordinated universal time) of the current transaction based on the system clock, and it assigns the value for the SysEndTime column to the maximum 9999-12-31. This marks the record as open.
- **UPDATES**: On an UPDATE, the system stores the previous value of the record in the history table and sets the value for the SysEndTime column to the UTC time of the current transaction based on the system clock. This marks the record as closed, with a period recorded for which the record was valid. In the current table, the record is updated with its new value, and the system sets the value for the SysStartTime column to the UTC time for the transaction based on the system clock. The value for the updated record in the current table for the SysEndTime column remains the maximum value of 9999-12-31.
- **DELETES**: On a DELETE, the system stores the previous value of the record in the history table and sets the value for the SysEndTime column to the UTC time of the current transaction based on the system clock. This marks the record as closed, with a period recorded to indicate when the record was valid. In the current table, the record is removed. Queries of the current table will not return this value. Only queries that deal with history data return data when a record is closed.
- **MERGE**: On a MERGE, MERGE behaves as an INSERT, an UPDATE, or a DELETE based on the condition for each record.

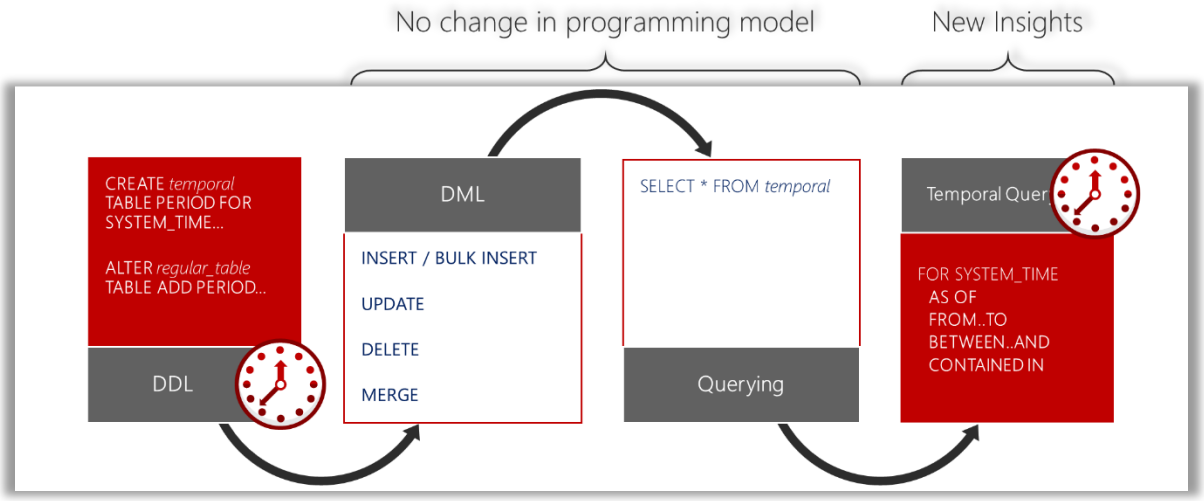


Figure 10: Temporal table on SQL Server 2016

Querying of current data does not differ from the non-temporal case, and it does not introduce any performance overhead either. There are several modes of querying historical data with the FOR SYSTEM_TIME clause. During the processing of query with FOR SYSTEM_TIME, SQL Server transparently adds a union between the current and history tables, and it propagates temporal predicates to filter data based on their period of validity (Figure 11).

For more information: Temporal tables

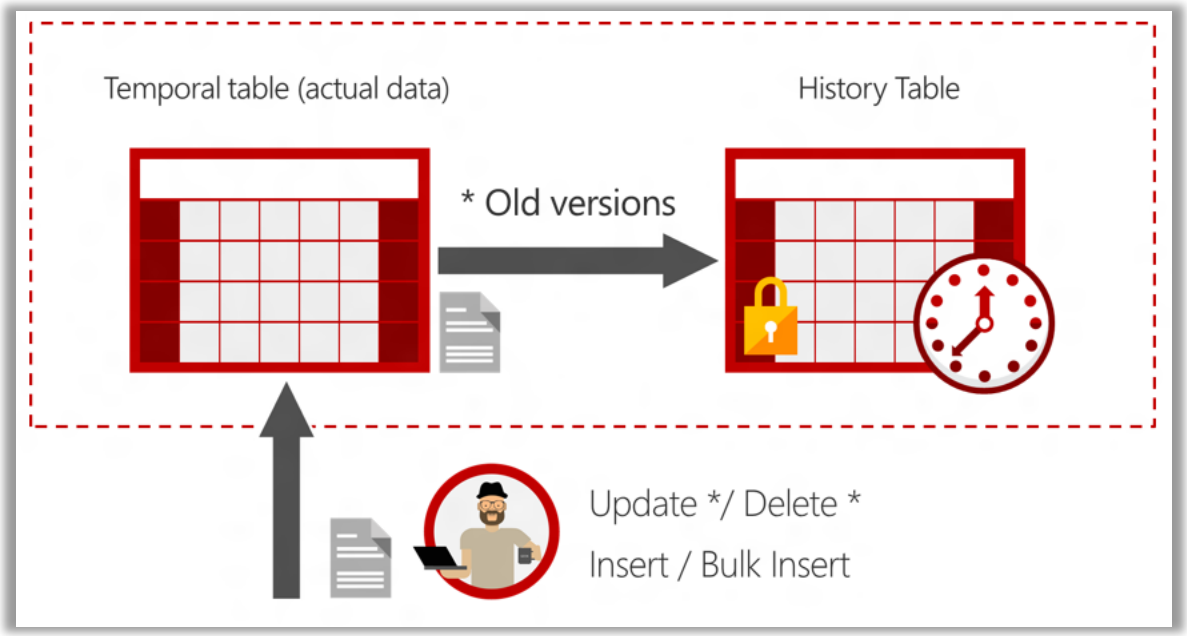


Figure 11: System-versioned temporal table

Security

Paramount among those areas where SQL Server users expect dependable, ever-evolving excellence—and where Microsoft continues to deliver—is security and privacy protection. SQL Server 2016 introduces several security innovations for your mission-critical applications.

Secure by default: lowering vulnerability

Microsoft and the SQL Server team take security seriously. More than a decade ago, Microsoft implemented the Trustworthy Computing initiative. The initiative requires SQL Server engineers to undergo regular security training and carry responsibility for security across their job duties, regardless of the work group to which they belong. This company-wide discipline to protect security and privacy was developed to create software that is secure by design—and to reduce by default the overall risks related to security. To that end, according to the National Institute of Standards and Technology (NIST) public security board, SQL Server has the lowest number of security vulnerabilities across the major database

vendors. In addition, the Information Technology Industry Council (ITIC)[1] has deemed SQL Server "the most secure database."

New in SQL Server 2016

SQL Server 2016 introduces several security innovations. As its name implies, Always Encrypted adds the unique capability of having data encrypted while at rest and in use, and the ability to query that data while it is encrypted. This is optimal for internal compliance, especially in regulated industries or for handling very sensitive data, and it is accomplished with minimal overhead.

Row-Level Security is another new feature that provides fine-grained access control over rows in a table based upon conditions the user sets up. Dynamic Data Masking is also new to SQL Server 2016, allowing one to obfuscate sensitive data fields in query results. A few other enhancements, like Transparent Data Encryption for In-Memory OLTP and enhanced auditing capabilities, give users new options for meeting security needs.

Always Encrypted

Always Encrypted is a new feature in SQL Server 2016 that protects data both at rest and in use (and keeps it encrypted in memory). By contrast, traditional encryption is applied only to data at rest, leaving data at risk when in use. Always Encrypted protects the data from rogue administrators and backup thieves.

Always Encrypted is a feature designed to protect sensitive data, such as credit card numbers or national identification numbers (U.S. Social Security numbers, for example), stored in SQL Server databases. Always Encrypted allows clients to encrypt sensitive data inside client applications without ever revealing the encryption keys to SQL Server. As a result, Always Encrypted provides a separation between those who own the data (and can view it) and those who manage the data (but should have no access to it). In essence, Always Encrypted removes the database and SQL Server from the attack surface area.

How it works

An Always Encrypted-enabled driver installed on the client computer automatically encrypts and decrypts sensitive data in the SQL Server client application. The driver encrypts the data in sensitive columns before passing the data to SQL Server, and it automatically rewrites queries to preserve the semantics of the application. Similarly, the driver transparently decrypts data that is stored in encrypted database columns and contained in query results. The following details explain how these features work (also illustrated in Figure 12):

- Users specify individual columns of particular tables to be encrypted.
- Once encrypted, the data appears as an encrypted binary collection at all stages within the SQL Server database: on disk, in-memory, and during computations.
- Client applications use a key (CMK) obtained from a trusted key store.

[1] Information Technology Intelligence Corp. (ITIC), SQL Server Delivers Industry-Leading Security, September 2012.

- Both encryption and decryption are done by the client driver, such as ADO.NET, JDBC, or ODBC. This driver will require access to the encryption key (via a trusted store) and, thereafter, will communicate with the SQL Server directly to effect transparent encryption.
- Specifically, when queries are parameterized, SqlClient will handshake with the SQL Server and identify which parameters are encrypted. It will manage this process in both directions.
- For example: The client code specifies a select statement with the parameter "where SSN = @SSN" and provides the parameter value "@SSN='123-45-6789'" and the driver itself intercepts the parameter value and properly encrypts it.

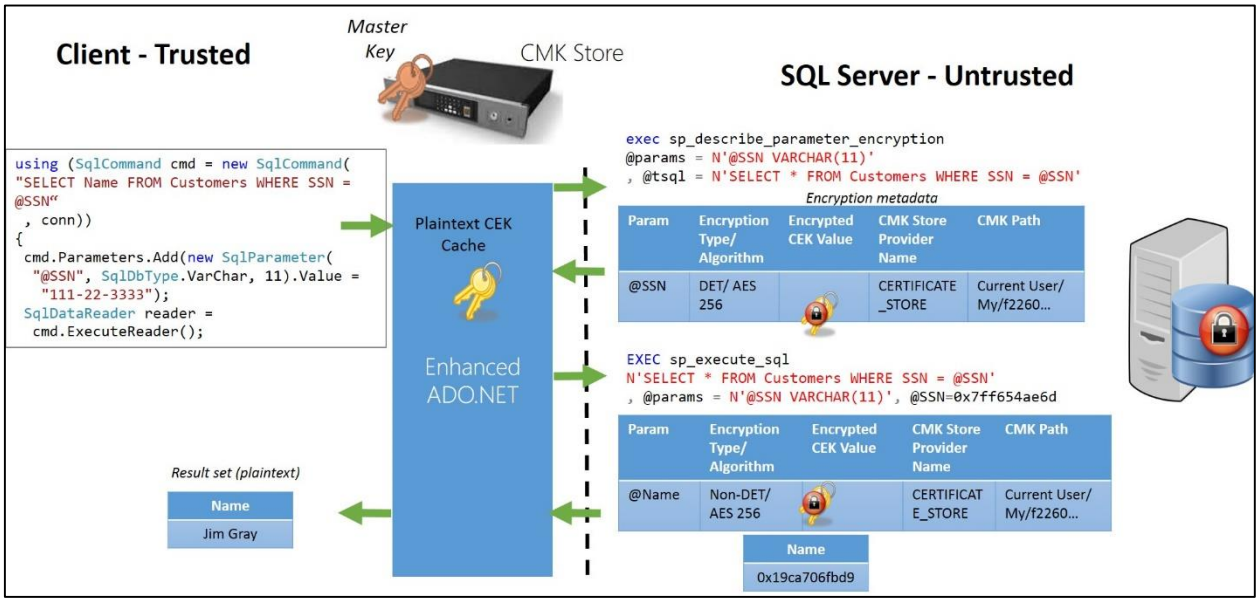


Figure 12: Queryable encryption with Always Encrypted

With existing applications, the setup also requires encrypting the (previously) plain text data in the selected columns. This can be accomplished in two ways:

- Using the Always Encrypted Wizard in SQL Server Management Studio, existing data columns can be encrypted or decrypted. This wizard will also manage key provisioning for encrypted columns. For more information: Always Encrypted Wizard
- If you are migrating data into a new database or table, setting "Column Encryption Setting" to "Enabled" within the Import/Export allows the migrating of data from plain text into an encrypted column.
  For more information: Column Encryption Setting

Two kinds of encryption methods are available in SQL Server 2016:

- **Randomized encryption** uses a method that encrypts data in a less predictable manner. Randomized encryption is more secure, but it prevents equality searches, grouping, indexing, and joining on encrypted columns.
- **Deterministic encryption** always generates the same encrypted value for any given plain text value. Using deterministic encryption allows grouping, filtering by equality, and joining tables based on encrypted values. However, it can also allow unauthorized users to guess information about encrypted values by examining patterns in the encrypted column.

Dynamic Data Masking

Dynamic Data Masking (DDM), new to SQL Server 2016, provides a mechanism to limit the exposure of sensitive data by controlling how the data appears in the output of database queries. Masking rules can be defined on particular database columns, indicating what type of masking will be applied when those columns are queried. Thus, the sensitive data is masked out when queried by users who don't have a business need to see that data.

Dynamic Data Masking can be used to obfuscate data in an application so that restricted data is not revealed. For example, a call center application that is used by a representative to identify a customer's payment information should be exposed only to the last four digits of the customer's credit card number. A simple DDM rule can ensure that any time the credit card number field is queried by the application, the number will be masked out except for the last four digits (Figure 13).

Another scenario in which to apply DDM is enabling engineers or operations personnel to access a production database for maintenance or troubleshooting purposes, without exposing the personally identifiable information (PII) in the database. A DDM policy can be enabled to obfuscate this data in real time for such a scenario, so that such tasks can be achieved without violating privacy policies.

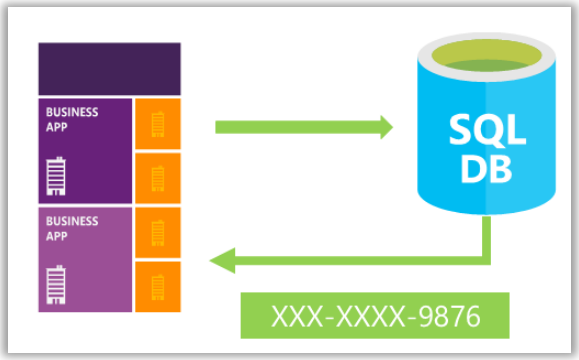


Figure 13: Dynamic Data Masking

How it works

A DDM rule can be defined on a particular column, indicating how that column will be masked when queried. There are no physical changes to the data in the database itself, so the data remains intact and fully available to authorized users or applications. Database operations also remain unaffected, and the masked data has the same data type as the original data. This means that DDM can often be applied without making any changes to database procedures or application code.

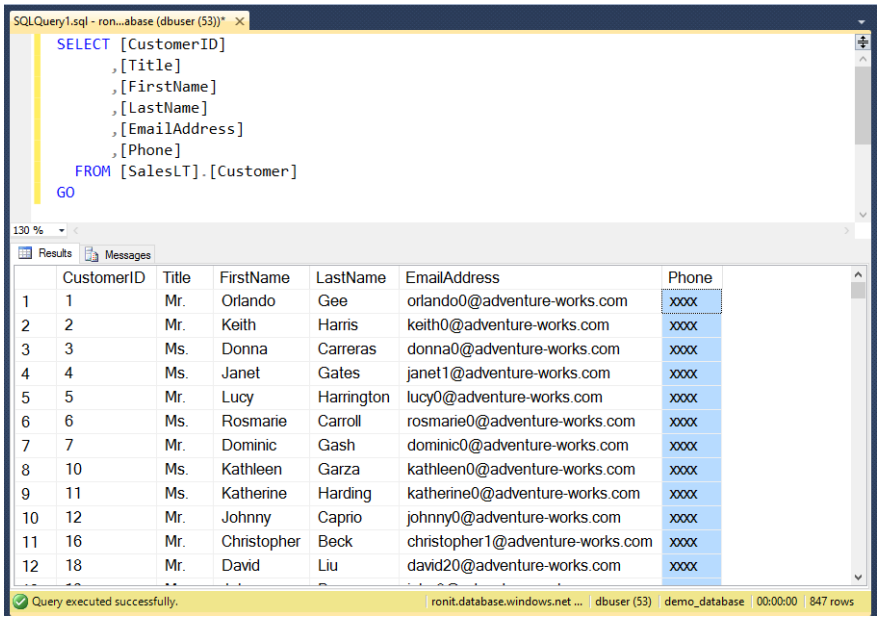
To add a data mask on a certain column in your database, simply alter that column by adding a mask and specifying the required masking type. The four masking types are described below:

- **Default**: To fully mask out the original value
- **Partial**: To specify which part of the data to expose
- **Random**: To replace the numeric value with a random value within a specified range
- **Email**: To expose the first character and keep the email format

Configure a masking function:

```
ALTER TABLE [SalesLT].[Customer]
ALTER COLUMN [Phone] ADD MASKED WITH (FUNCTION = 'default()')
```

Results:



	CustomerID	Title	FirstName	LastName	EmailAddress	Phone
1	1	Mr.	Orlando	Gee	orlando0@adventure-works.com	xxxx
2	2	Mr.	Keith	Harris	keith0@adventure-works.com	xxxx
3	3	Ms.	Donna	Carreras	donna0@adventure-works.com	xxxx
4	4	Ms.	Janet	Gates	janet1@adventure-works.com	xxxx
5	5	Mr.	Lucy	Harrington	lucy0@adventure-works.com	xxxx
6	6	Ms.	Rosmarie	Carroll	rosmarie0@adventure-works.com	xxxx
7	7	Mr.	Dominic	Gash	dominic0@adventure-works.com	xxxx
8	10	Ms.	Kathleen	Garza	kathleen0@adventure-works.com	xxxx
9	11	Ms.	Katherine	Harding	katherine0@adventure-works.com	xxxx
10	12	Mr.	Johnny	Caprio	johnny0@adventure-works.com	xxxx
11	16	Mr.	Christopher	Beck	christopher1@adventure-works.com	xxxx
12	18	Mr.	David	Liu	david20@adventure-works.com	xxxx

Privileged users are exempt from masking, so they will always get the real data when performing queries. By granting certain users the UNMASK permission, they can be assigned privileges to access the actual data.

Row-Level Security

Applications often need to limit a user's access to only certain rows of data in a table. For example, an oil and gas exploration application might restrict access to oil-well production data based on an analyst's region and role. A healthcare application might restrict access to patient data based on a doctor's staffing assignments. Similarly, a multitenant application with a "shared database, shared schema" tenancy model needs to prevent tenants from accessing rows of data that do not belong to them.

SQL Server 2016 introduces a built-in Row-Level Security (RLS) feature that enables developers and DBAs to implement fine-grained access control over rows in a table. Using RLS, data can be stored for different customers, departments, or tenants in the same table, while restricting read and write access to rows based on criteria defined by DBAs. For example, rows returned by “SELECT * FROM myTable” could be filtered according to the identity of the logon user, the current user's role or group memberships, or even the value of a session-scoped variable like SESSION_CONTEXT. Another example would be using a block predicate with an INSERT statement to prevent the addition of data under an incorrect department or tenant.

RLS works transparently at query time, with no application changes required. By centralizing access logic within the database, administrators can maintain a consistent data access policy and reduce the risk of accidental data leakage. Implementing RLS in the database can greatly reduce client application maintenance and complexity.

How it works

RLS is a form of predicate-based access control, which means it limits access by automatically applying a security predicate to all queries on a table. This predicate determines which users can access which rows. A simple security predicate might be WHERE SalesRepName = USER_NAME(). A more complicated security predicate could include joins to look up information in other tables, such as an employee's organizational position.

RLS supports two types of security predicates:

- **Filter predicates** silently filter SELECT, UPDATE, and DELETE operations to exclude rows that do not satisfy the predicate.
- **Block predicates** explicitly block INSERT, UPDATE, and DELETE operations that do not satisfy the predicate.

In order to control both read and write access to rows, both types of predicates should be used on the same table.

To enable RLS on a table, users should first create an inline table-valued function that defines their access criteria. Then, using this function, a security policy is created that adds filter and block predicates on the tables. A security policy is a collection of security predicates for managing row-level security across multiple tables—for example, an account policy that applies multiple security predicates to account-related tables, and a human resources policy that applies several security predicates to various HR tables.

Flexible access criteria

RLS integrates seamlessly with SQL Server's built-in security system of users and roles. Security predicates can limit access based on the querying principal, using functions like USER_NAME() or DATABASE_PRINCIPAL_ID(). Access can also be limited based on the querying principal's SQL role or Windows group memberships, using functions like IS_MEMBER().

For applications that use a single logon to connect to the database (such as a service account), users can limit access based on values set by the application in the SESSION_CONTEXT. This approach is common for web applications, in which all end users share the same SQL logon in order to enable efficient connection pooling. Upon opening a connection from the pool, the application uses sp_set_session_context to set a read-only key-value pair for the current user ID. Making the value read-only for the session helps prevent SQL injection attacks from being able to elevate to a different user. RLS security predicates can then use the user ID stored in the SESSION_CONTEXT to control access to rows.

Enhanced in SQL Server 2016

Built-in tools for enabling compliance: SQL Server audit tools

Auditing an instance of the SQL Server Database Engine or an individual database involves tracking and logging events that occur on the Database Engine. SQL Server Audit lets users create server audits (which can contain server audit specifications for server-level events) and database audit specifications for database-level events. Audited events can be written to the event logs or to audit files. All editions of SQL Server support server-level audits. Database-level auditing is limited to the enterprise, developer, and evaluation editions of SQL Server.

In order to meet the auditing requirement, SQL Server 2016 provides the following features:

- **User-defined audit** allows the middle-tier application to write custom events into the audit log, which enables more flexibility to store audit information. As part of an auditing policy, users can create an audit logon based on a particular query or event. The event will be triggered whenever a condition or query is encountered or satisfied.
- **Audit filtering** provides greater flexibility to filter wanted events in an audit log. As part of an auditing policy, administrators need to be able to log which clients are accessing selected tables that contain sensitive data. This is done by creating an audit filter that will record the table, user, date, and time when the table was accessed.
- **Audit resilience** gives the ability to recover auditing data from temporary file and network issues to help ensure that audit logs are not lost during failover. The resilience features implemented in SQL Server Audit means that the audit logging is now tolerant to loss of connectivity to the target directory and will recover automatically once the network connection is re-established. Users can implement a policy for processing to continue in the event of an audit log failure. The audit process is also able to record that auditing has been paused or stopped.

Support for Azure Key Vault

SQL Server now supports the management of encryption keys in Azure Key Vault (AKV) for enterprise editions. Azure Key Vault offers central key management, leverages hardware security modules (HSMs), and promotes the separation of key management from the management of data to help meet regulatory compliances.

SQL Server's support for AKV is available through the SQL Server Connector for AKV for all enterprise versions of SQL Server starting with 2012 through the 2016 release. The connector is a downloadable dynamic-link library (DLL) file that serves as an Extensible Key Management (EKM) provider for SQL Server. This connector is especially important to those using SQL Server Virtual Machines who want to leverage AKV for managing their encryption keys. SQL Server Virtual Machines allow for quick deployment of SQL Server and is ideally suited for re-hosting existing SQL Server applications in the cloud or for extending portions of database management operations into the cloud. With AKV integration, both on-premises and SQL Server-in-a-VM users can assume control of encryption keys for Transparent Data Encryption (TDE),

column-level encryption (CLE), and backup encryption while leveraging the additional security benefits of Azure Key Vault.

Transparent Data Encryption

SQL Server Transparent Data Encryption (TDE), exclusive to SQL Server Enterprise edition, allows organizations to encrypt data when it is stored on a disk and decrypt it when it is read into memory. This enables software developers to encrypt database files, log files, and backup files without changing existing applications.

TDE has been updated to take advantage of the latest Intel AES-NI hardware acceleration found in most Intel processors over the last several years. This results in significantly faster performance. Following initial encryption, there is little CPU impact for many workloads.

TDE now also supports storage of memory-optimized OLTP tables. This allows for greater security, along with the performance enhancements provided by memory optimization.

Backup encryption

SQL Server encrypts data while creating a backup. By specifying the encryption algorithm and the encryptor (a certificate or asymmetric key) when creating a backup, you can create an encrypted backup file. On-premises and Windows Azure Storage locations are supported for this process. In addition, encryption options can be configured for SQL Server Managed Backup to Windows Azure operations, a new feature introduced in SQL Server 2014. To encrypt during backup, you must specify an encryption algorithm and an encryptor to secure the encryption key. The following are the supported encryption options:

- **Encryption algorithm**. The supported encryption algorithms are AES 128, AES 192, and AES 256.
- **Encryptor**. A certificate or asymmetric key.

Backup encryption is now supported with compression and will automatically use Intel AES-NI hardware acceleration.

Audit enhancements

SQL Server 2016 can now audit the success or failure of operations. When a user attempts to perform an operation, the user must first have permission to perform that operation. Prior versions of SQL Server could track and audit whether a permissions check succeeded or failed. But even if a user has permission, it is still possible that the desired operation might fail due to referential integrity rules or another constraint. SQL Server 2016 audits transaction events, both explicit as well as implicit, so that the success or failure of the operation (in addition to the permission check) can be determined from the audit log.

Controlled access to data across business intelligence tools: Microsoft SharePoint and Microsoft Active Directory

As Microsoft continues to deliver business intelligence tools that are used by a broader set of users, security concerns also increase because of broader implications if security is compromised. SQL Server helps organizations secure end-user data analytics with built-in IT controls, including new SharePoint and Active Directory security models for end-user reports that are published and shared in SharePoint. Enhanced security models provide control at row and column levels.

Availability

SQL Server high availability solutions provide mission-critical uptime, fast failover, improved manageability, and better use of hardware resources. Features like SQL Server AlwaysOn continue to get better and more powerful with every release.

High availability of mission-critical systems

SQL Server 2016 adds significant new enhancements to AlwaysOn features. You can now have up to three synchronous replicas in an availability group, as well as readable asynchronous replicas. These replicas can exist in different domains and be on-premises and on Azure virtual machines. SQL Server 2016 also adds better load balancing of replicas using a round-robin methodology. This version has also resolved compatibility issues with Distributed Transaction Coordinator (DTC) and SQL Server Integration Services (SSIS) when using AlwaysOn. You now also have greater uptime with Enhanced Online Operations when conducting ALTER or TRUNCATE operations on tables. SQL Server 2016 also offers the high availability and scalability capabilities of SQL Server standard edition by adding AlwaysOn basic for two-node failover and removing core and memory restrictions on the standard edition.

AlwaysOn

AlwaysOn was first introduced in SQL Server 2012, and SQL Server 2016 continues to deliver an enhanced high availability solution (Figure 14). This integrated high availability and disaster recovery solution provides redundancy within a datacenter and across datacenters to help enable fast failover of applications during planned and unplanned downtime. AlwaysOn delivers a suite of capabilities rolled into a single solution.

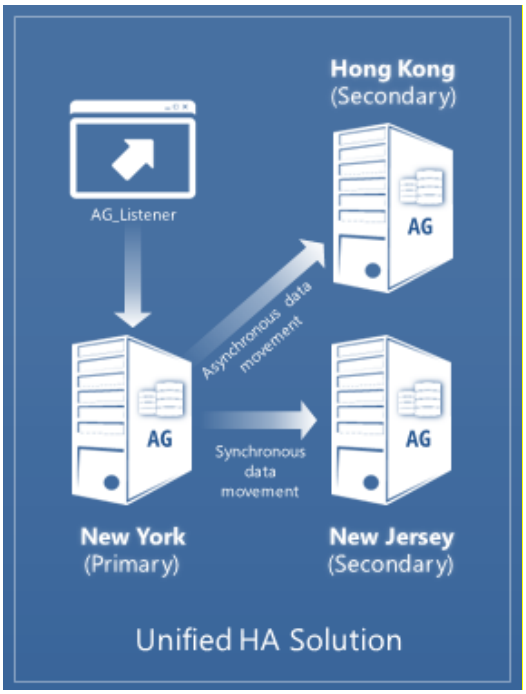


Figure 14: Unified high availability solution

Availability Groups is a high availability and disaster recovery solution that provides an enterprise-level alternative to database mirroring. Availability Groups is an integrated set of options that includes automatic and manual failover of a group of databases, support for as many as eight secondary replicas (secondaries), faster failover for applications, and automatic page repair. Each availability group is a container for a discrete set of user databases known as availability databases that do failover together. An availability group can have many possible failover targets (secondary replicas). Moreover, organizations can easily configure secondary replicas to support read-only access to secondary databases and backup secondary databases. The addition of Availability Groups removes the requirement of shared disk storage, such as storage area network (SAN) or network-attached storage (NAS), for deployment of a Failover Cluster Instance.

Availability Groups Listener enables faster failover in client connections for AlwaysOn in scenarios that employ multiple subnets. Client applications can now achieve failover across multiple subnets (as many as 64) almost as fast as they can achieve failover within a single subnet. Meanwhile, the ability to set the connection from within applications to read-only (instead of read and write) empowers organizations to control the type of workloads running on their high availability servers, so they can more efficiently manage their resources.

Failover Cluster Instances enhances SQL Server Failover Clustering and supports multisite clustering across subnets, which helps enable failover of SQL Server instances across datacenters. Faster and more predictable failover of instances is another key benefit that helps ensure faster recovery of applications. By supporting Windows Server Cluster Shared Volumes, AlwaysOn further improves use and management of SAN storage through increased resilience of storage failover and avoidance of the drive-letter limitation in SAN.

Multiple, Active Secondaries enables use of as many as eight secondary instances for running report queries (many times faster than replication) and backup operations, even in the presence of network failures—which helps in repurposing idle hardware and improving resource utility. It also helps to dramatically improve performance for both primary and secondary workloads because they are no longer competing for resources.

SQL Server AlwaysOn to Azure Virtual Machine enables organizations to add secondary replicas in an Azure Virtual Machine through the Add Azure Replica Wizard. They can then use this replica for disaster recovery, reporting, and backup operations. This configuration can lower capital expenses by eliminating the need to purchase additional hardware for AlwaysOn secondaries.

New in SQL Server 2016

Basic Availability Groups

Now available in SQL Server 2016 Standard edition or higher, Basic Availability Groups provides failover support for a single database, replacing the database mirror feature of previous versions. Basic availability groups enable a single replica for a primary database, using either synchronous or asynchronous commit mode. This secondary replica remains inactive unless a failure occurs in the primary. Basic availability groups support hybrid environments, spanning on-premises and Azure.

For more information: AlwaysOn Basic Availability Groups

Enhanced in SQL Server 2016

Distributed Availability Groups

AlwaysOn supports distributed Availability Groups, which are "loosely coupled" groups. Secondary replicas of an Availability Group can exist in different geographical regions than the primary. This enables serving read-only workloads for remote regions, as well as disaster recovery support.

Automatic replica seeding

Traditionally, adding a replica requires seeding the new replica with data from a recent backup of the primary database. The Add Azure Replica Wizard adds a replica of your databases to Azure Blob Storage. In SQL Server 2016, the group listener is created and configured within the wizard. This allows clients to connect seamlessly to the Azure replica after failover, as soon as the wizard completes its setup and without additional complex steps.

Distributed transactions

AlwaysOn Availability Groups supports distributed transactions and the Microsoft Distributed Transaction Coordinator (MSDTC) on Windows Server 2016. This applies to distributed transactions between databases hosted by two different SQL Server instances. It also applies to distributed transactions between SQL Server and another DTC-compliant server.

Automatic failover replicas

SQL Server 2016 supports automatic failover to synchronous-commit replicas. Availability Groups can be configured to do a failover to a secondary replica automatically, so that the designated secondary becomes the group primary, and the former primary switches to the secondary role. Since automatic failover mode requires synchronous-commit replicas, failover occurs without data loss.

Load balancing

For scalability, SQL Server 2016 adds load balancing of readable secondaries capabilities. This allows you to define one or more groups of readable secondaries to load balance. Now you can configure the read-only routing (ROR) lists to round-robin among a specific set of secondaries (for each primary). Connections are assigned round-robin to members of the group. This is set up by specifying an extension of the READ_ONLY_ROUTING list (Figure 15). This version also increases the number of auto-failover targets from two to three for additional synchronous failover targets. You are now able to specify up to three total secondaries for automatic failover.

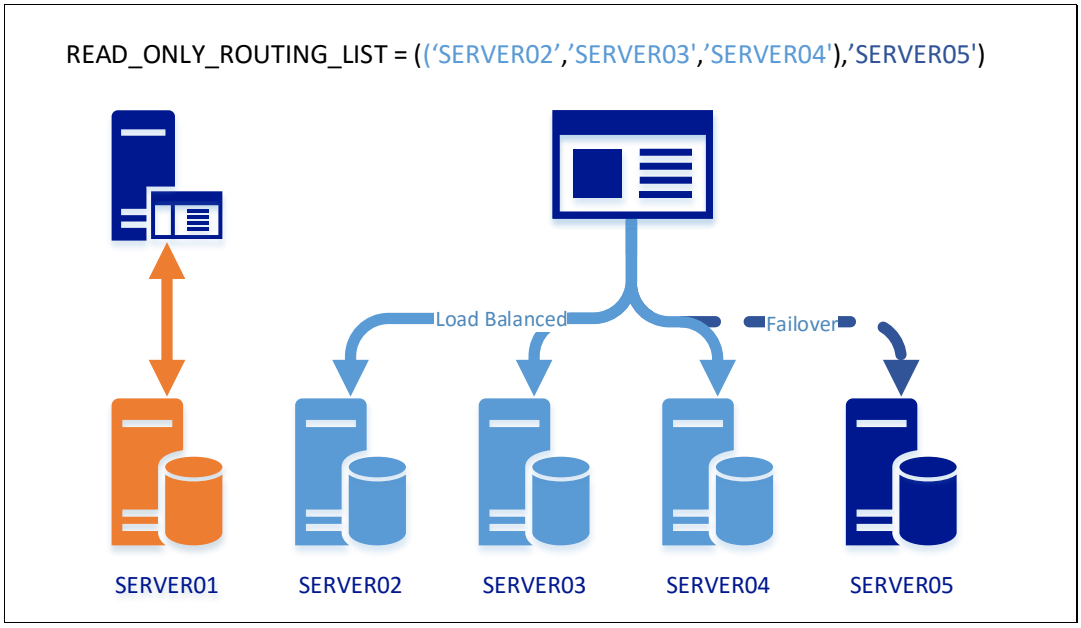


Figure 15: AlwaysOn readable secondary load balancing

Manageability

Manageability has also improved in several areas within SQL Server 2016, including support for MSDTC-enrolled transactions for Availability Groups databases. MSDTC resources are tied to the database instead of the instance so, on failover, the Distributed Transaction Coordinator sees the same resource on the new primary, and transaction outcomes can be resolved. In previous versions of SQL Server, MSDTC is not supported for Availability Groups databases. But now, with SQL Server 2016, it will be fully supported. MSDTC will also require an operating system and the most recent version of Windows Server for full support across all scenarios.

Group Managed Service Accounts (GMSAs) are now supported for AlwaysOn failover clusters. GMSAs are domain-level accounts that provide automatic password management and simplified service principal name (SPN) management. When GMSA are used as service principals, the operating system manages the account password, forgoing the need for an administrator to log on to multiple machines to update a password.

In Windows Server 2012 R2 and previous versions, a cluster can only be created between member nodes joined to the same domain. Windows Server 2016 Technical Preview breaks down these barriers and introduces the ability to create a Failover Cluster without Active Directory dependencies. Now with this support from Windows Server 2016, you can use SQL Server Availability Groups in conjunction with the Windows Failover Cluster independent of domain membership or trust, including support for workgroups.

Online database operations

SQL Server continues to enable organizations to achieve high availability during resource-intensive operations. For example, the ability to rebuild online indexes in a single partition provides partition-level control for users who need continual access to the database. This approach also requires fewer resources (CPU and memory), so it minimizes the impact of rebuilding indexes. Specifically, the ability to manage priorities for locking tables gives organizations greater control over the impact of maintenance operations on running transactions—from table switching to online index rebuild operations—by allowing database administrators to specify whether or not to terminate processes that block their ability to lock tables.

Enhanced online operations

With SQL Server 2016, you can rebuild online indexes in a single partition, and this provides partition-level control for users who need continual access to the database. Also, SQL Server 2016 allows database administrators to specify whether or not to terminate processes that block their ability to lock tables. SQL Server 2016 now provides 100 percent uptime, with enhanced online database operations when conducting ALTER or TRUNCATE operations on tables.

Predictable, efficient, and flexible data backups

Managed Backup

In SQL Server 2016, Managed Backup to Microsoft Azure uses the new block blob storage for backup files. This allows the option to stripe backup sets, enabling backup-file sizes up to 12.8 TB. Other changes and enhancements to Managed Backup include:

- System databases can be backed up with managed backups.
- Databases in full, bulk logged, and simple recovery model are supported.
- Both automated and custom scheduling of backups are supported.

Database Recovery Advisor (SQL Server Management Studio)

The Database Recovery Advisor facilitates the construction of restore plans that implement optimal correct restore sequences. SQL Server Management Studio has addressed customer concerns around

many known database restore issues and enhancements. Major enhancements to the Database Recovery Advisor include:

- **Restore-plan algorithm**. The algorithm used to construct restore plans has improved significantly, particularly for complex restore scenarios. Many edge cases, including forking scenarios in point-in-time restores, are handled more efficiently than in previous versions of SQL Server.
- **Point-in-time restores**. The Database Recovery Advisor greatly simplifies restoring a database to a given point in time. A visual backup timeline significantly enhances support for point-in-time restores. This visual timeline allows you to identify a feasible point in time as the target recovery point for restoring a database. The timeline facilitates traversing a forked recovery path (a path that spans recovery forks). A given point-in-time restore plan automatically includes the backups that are relevant to restoring to your target point in time (date and time).

  For more information: Restore a SQL Server Database to a point in time (full recovery model)

Scalability

The interaction between Microsoft SQL Server and Microsoft Windows Server is an area that can lead to large improvements in scalability. With SQL Server and Windows Server, physical processing now scales up to 640 logical processors, and virtual machines scale up to 64 logical processors. SQL Server also uses storage spaces and network virtualization to optimize your resources. It can run on Windows Server Core to lower the surface area of attack. From advancements in compute, to storage and networking, all have a direct impact on mission-critical SQL Server workloads. A number of other tools and techniques can help boost scalability as well.

Setting up a private cloud can be a complex undertaking. SQL Server has several enhancements that can make this job easier. Additionally, SQL Server is integrated into the Microsoft Cloud Platform System, discussed below, which provides new possibilities for scaling a private cloud very quickly.

Support for Windows Server Core

SQL Server is supported on Windows Server Core—the Windows Server edition with the smallest footprint. Because Windows Server Core requires less maintenance and fewer operating system (OS) patches, planned downtime is greatly reduced when you run SQL Server on Windows Server Core. The percentage reduction in patching and OS reboots can be as much as 50 to 60 percent in certain environments, depending on the server roles that are enabled and the type of patches that are applied.[2]

Faster live migration

Windows Server allows simultaneous migration of any number of SQL Server virtual machines you need, which helps organizations maintain the availability of SQL Server while decreasing planned downtime. Faster live migration also helps organizations decrease planned downtime by allowing migration of many SQL Server virtual machines (using priority settings) in a clustered environment, and by using as much as 10 GB of network bandwidth.

---

[2] "Why Is Server Core Useful," Microsoft TechNet, http://technet.microsoft.com/en-us/library/dd184076.aspx, accessed May 15, 2013.

Live migration for non-clustered virtual machines

Windows Server allows live migration of SQL Server virtual machines in a non-clustered environment, both in centrally shared and non-shared virtual machine storage scenarios. This practice helps organizations reduce the cost and complexity of SQL Server deployments in virtualized environments while maintaining availability during planned downtime.

Cluster-Aware Updating

With Cluster-Aware Updating, organizations can apply updates automatically to the host operating system—or to other system components in a clustered SQL Server environment—while maintaining availability. This approach can significantly help increase SQL Server availability during the update process in both virtualized and non-virtualized environments.

Dynamic Quorum

Windows Server Failover Clustering Dynamic Quorum enables the SQL Server AlwaysOn cluster to dynamically adjust the number of quorum votes that are required to keep the system running. This adjustment can simplify setup by as much as 80 percent. It also helps increase availability of a SQL Server cluster in failover scenarios in both virtualized and non-virtualized environments—with the ability to recalculate a quorum as needed and still maintain a working cluster.

Enhanced in SQL Server 2016

Hardware acceleration for encrypting and decrypting transparent data

Transparent Data Encryption (TDE) has been enhanced with support for Intel AES-NI hardware acceleration of encryption. This will reduce the CPU overhead of turning on TDE. SQL Server 2016 now implements Microsoft's next-generation cryptography (CNG) API with support for specialized microprocessor instructions (such as Intel's AES-NI, which is an extension to the x86 instruction set available from Intel and AMD) in order to speed up encryption/decryption of data for TDE-using hardware. In the case of Intel, the hardware acceleration is advertised to improve performance of an implementation of AES (Advanced Encryption Standard) by three to 10 times over a pure software implementation.

Parallelizing the decryption built-in function to improve read performance

Before SQL Server 2016, the decryption function was marked as sequential and, as a result, it was not parallelizable by the optimizer when SQL Server is running on a multi-core machine. The lack of parallelism could be an issue in some situations. SQL Server 2016 marks the decryption function as parallelizable, which improves the performance of read operations on encrypted data (such as SQL Server column encryption). Allowing the decryption functions to run in parallel should result in dramatically better response times for queries with encrypted data columns.

Enhancements working with Windows Server 2016

SQL Server 2016 is able to make use of several enhancements in Windows Server 2016, such as increased upward limits on memory available to the operating system.

Buffer pool extension

SQL Server enables improvement to query performance by allowing the use of non-volatile devices such as solid-state drives (SSDs) to reduce SQL Server memory pressure with no risk of data loss. The configuration is simple and can greatly improve query performance.

Tier-1 partitioning: scale to 15,000 partitions

SQL Server supports as many as 15,000 table partitions. This support enables large "sliding window" scenarios. This means that applications such as SAP (which take tens of thousands of snapshots of data in daily or hourly partitions) can significantly extend the length of time data is held before it is "pushed out" to allow for new data to enter—generally making it easier to manage these large amounts of data. This capability also helps administrators streamline maintenance of large data sets within file groups that need data switched in and out according to the needs of the data warehouse.

Scalable real-world application testing: Distributed Replay

Organizations need a way to apply real-world application loads to their applications within test environments. Previously, they could use Microsoft SQL Server Profiler, which only allowed simulation of a workload from a single computer. This limitation made it difficult to test large-scale workload simulations. SQL Server Distributed Replay helps organizations simplify application testing and minimize errors with application changes, configuration changes, and upgrades. This multithreaded replay utility enables simulation to test production workload scenarios after upgrades or configuration changes—ultimately leading to protected performance during changes. Additionally, integration with Microsoft SQL Server Upgrade Assistant can help organizations assess the impact of future SQL Server upgrades.

TempDB optimization

SQL Server also allows you to scale up your database with enhanced data caching, using support for multiple TempDB files per instance for multi-core environments. This reduces both metadata contention and allocation contention for TempDB workloads, improving performance and scalability.

Core Engine Scalability improvement

Available in SQL Server 2016 is an exciting Core Engine Scalability improvement that allows you to dynamically partition thread-safe memory objects by non-uniform memory access (NUMA) node or by CPU. This improvement will enable higher scalability of high-concurrency workloads running on NUMA hardware. Thread-safe memory objects (of type CMemThread) will be dynamically promoted to be partitioned by NUMA node or by CPU based on workload characteristics and contention factors. In SQL 2012 and SQL 2014, TF8048 is needed to promote memory objects that are partitioned by node to be partitioned by CPU. This improvement not only eliminates the need for the trace flag, but it also dynamically determines partition based on contention.

Conclusion

SQL Server delivers a new standard in enabling mission-critical operations with true enterprise-class performance, security, availability, and scalability built into the solution. SQL Server delivers mission-critical performance and scale with predictable performance across server activities, including complex queries, data integration, and analysis. Because SQL Server is designed to meet security standards, it has minimal total surface area and database software that is inherently more secure. Enhanced security, combined with built-in, easy-to-use tools and controlled data access, helps organizations meet strict compliance policies. Integrated high availability solutions enable faster failover and more reliable backups—and they are easier to configure, maintain, and monitor, which helps organizations reduce the total cost of ownership (TCO). SQL Server supports complex data types *and* non-traditional data sources, and it handles them with the same attention—so organizations experience seamless support for a variety of platforms and heterogeneous environments.

More information

More information about topics discussed in this white paper can be found on the SQL Server website.

Feedback

Did this paper help you? Please give us your feedback by telling us on a scale of 1 (poor) to 5 (excellent) how you rate this paper and why you have given it this rating. More specifically:

- Are you rating it highly because of relevant examples, helpful screen shots, clear writing, or another reason?
- Are you rating it poorly because of examples that do not apply to your concerns, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of the white papers we release.

Please send your feedback to: mailto:sqlsrvwpfeedback@microsoft.com