Monitoring performance by using the Query Store

10/26/2017 • 17 minutes to read • Contributors

In this article

Enabling the Query Store
Information in the Query Store
Use the Regressed Queries Feature
Finding wait queries
Configuration Options
Related Views, Functions, and Procedures
Key Usage Scenarios
See Also

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

The SQL Server Query Store feature provides you with insight on query plan choice and performance. It simplifies performance troubleshooting by helping you quickly find performance differences caused by query plan changes. Query Store automatically captures a history of queries, plans, and runtime statistics, and retains these for your review. It separates data by time windows so you can see database usage patterns and understand when query plan changes happened on the server. You can configure query store using the ALTER DATABASE SET option.

For information about operating the Query Store in Azure SQL Database, see Operating the Query Store in Azure SQL Database.

Enabling the Query Store

Query Store is not active for new databases by default.

FEEDBACK

Use the Query Store Page in SQL Server Management Studio

1. In Object Explorer, right-click a database, and then click **Properties**.

   > **Note**
   >
   > Requires at least version 16 of Management Studio.

2. In the **Database Properties** dialog box, select the **Query Store** page.

3. In the **Operation Mode (Requested)** box, select **On**.

Use Transact-SQL Statements

Use the **ALTER DATABASE** statement to enable the query store. For example:

```sql
ALTER DATABASE AdventureWorks2012 SET QUERY_STORE = ON;
```

For more syntax options related to the query store, see ALTER DATABASE SET Options (Transact-SQL).

> **Note**
>
> You cannot enable the query store for the **master** or **tempdb** database.

Information in the Query Store

Execution plans for any specific query in SQL Server typically evolve over time due to a number of different reasons such as statistics changes, schema changes, creation/deletion of indexes, etc. The procedure cache (where cached query plans are stored) only stores the latest execution plan. Plans also get evicted from the plan cache due to memory pressure. As a result, query performance regressions caused by execution plan changes can be non-trivial and time consuming to resolve.

Since the query store retains multiple execution plans per query, it can enforce policies to direct the query processor to use a specific execution plan for a query. This is referred to as plan forcing. Plan forcing in Query Store is provided by using a mechanism similar to the USE PLAN query hint, but it does not require any change in user applications. Plan forcing can resolve a query performance regression caused by a plan change in a very short period of time.

Wait stats are another source of information that helps to troubleshoot performance in SQL Server. For a long time, wait statistics were available only on instance level, which made it hard to backtrack it to the

actual query. In SQL Server 2017 and Azure SQL Database we added another dimension in Query Store that tracks wait stats.

Common scenarios for using the Query Store feature are:

- Quickly find and fix a plan performance regression by forcing the previous query plan. 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, assisting a DBA in troubleshooting performance resource problems.
- Identify top *n* queries (by execution time, memory consumption, etc.) in the past *x* hours.
- Audit the history of query plans for a given query.
- Analyze the resource (CPU, I/O, and Memory) usage patterns for a particular database.
- Identify top n queries that are waiting on resources.
- Understand wait nature for a particular query or plan.

The query store contains three stores:

- a **plan store** for persisting the execution plan information.
- a **runtime stats store** for persisting the execution statistics information.
- a **wait stats store** for persisting wait statistics information.

The number of unique plans that can be stored for a query in the plan store is limited by the **max_plans_per_query** configuration option. To enhance performance, the information is written to the stores asynchronously. To minimize space usage, the runtime execution statistics in the runtime stats store are aggregated over a fixed time window. The information in these stores is visible by querying the query store catalog views.

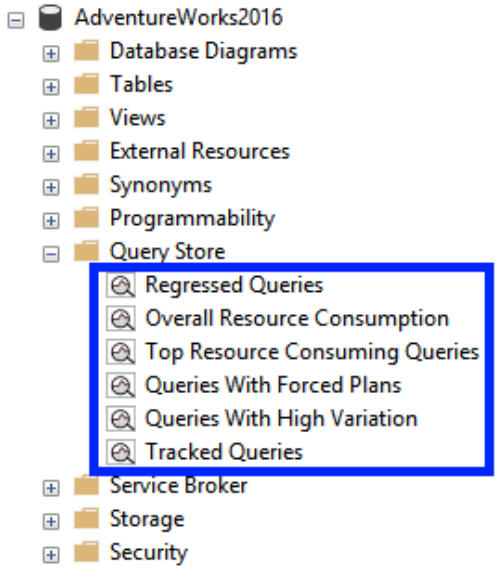
The following query returns information about queries and plans in the query store.

SQL Copy

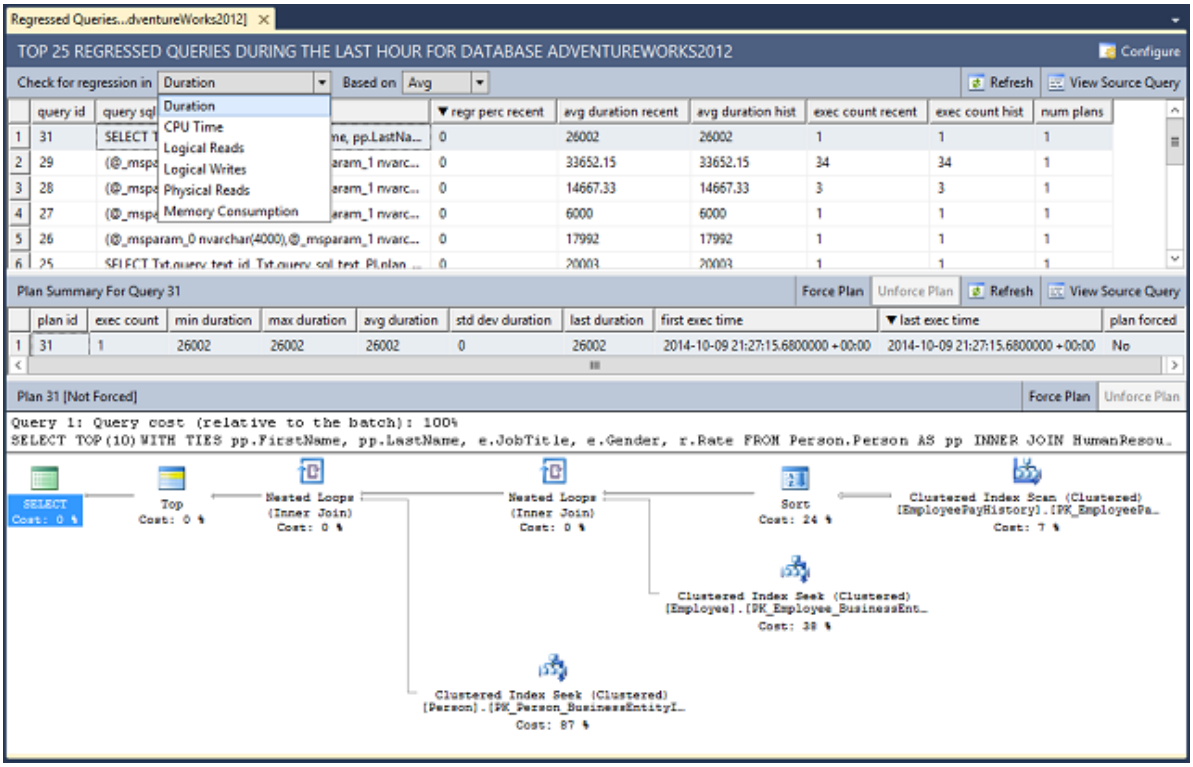
```sql
SELECT Txt.query_text_id, Txt.query_sql_text, Pl.plan_id, Qry.*
FROM sys.query_store_plan AS Pl
INNER JOIN sys.query_store_query AS Qry
    ON Pl.query_id = Qry.query_id
INNER JOIN sys.query_store_query_text AS Txt
    ON Qry.query_text_id = Txt.query_text_id ;
```

Use the Regressed Queries Feature

After enabling the query store, refresh the database portion of the Object Explorer pane to add the **Query Store** section.



Select **Regressed Queries** to open the **Regressed Queries** pane in Management Studio. The Regressed Queries pane shows you the queries and plans in the query store. Use the drop down boxes at the top to select queries based on various criteria. Select a plan to see the graphical query plan. Buttons are available to view the source query, force, and unforce a query plan, and refresh the display.



To force a plan, select a query and plan, and then click **Force Plan**. You can only force plans that were saved by the query plan feature and are still retained in the query plan cache.

Finding wait queries

Starting with SQL Server 2017 CTP 2.0 and SQL Database, wait statistics per query over time are available in Query Store. In Query Store, wait types are combined into **wait categories**. The mapping of wait categories to wait types is available in sys.query_store_wait_stats (Transact-SQL).

Wait categories are combining different wait types into buckets similar by nature. Different wait categories require a different follow up analysis to resolve the issue, but wait types from the same category lead to very similar troubleshooting experiences, and providing the affected query on top of waits would be the missing piece to complete the majority of such investigations successfully.

Here are some examples how you can get more insights into your workload before and after introducing wait categories in Query Store:

Previous experience	New experience	Action
High RESOURCE_SEMAPHORE waits per database	High Memory waits in Query Store for specific queries	Find the top memory consuming queries in Query Store. These queries are probably delaying further progress of the affected queries. Consider using MAX_GRANT_PERCENT query hint for these queries, or for the affected queries.
High LCK_M_X waits per database	High Lock waits in Query Store for specific queries	Check the query texts for the affected queries and identify the target entities. Look in Query Store for other queries modifying the same entity, which are executed frequently and/or have high duration. After identifying these queries, consider changing the application logic to improve concurrency, or use a less restrictive isolation level.
High PAGEIOLATCH_SH waits per database	High Buffer IO waits in Query Store for specific queries	Find the queries with a high number of physical reads in Query Store. If they match the queries with high IO waits, consider introducing an index on the underlying entity, in order to do seeks instead of scans, and thus minimize the IO overhead of the queries.
High SOS_SCHEDULER_YIELD waits per database	High CPU waits in Query Store for specific queries	Find the top CPU consuming queries in Query Store. Among them, identify the queries for which

high CPU trend correlates with high CPU waits for the affected queries. Focus on optimizing those queries – there could be a plan regression, or perhaps a missing index.

Configuration Options

The following options are available to configure query store parameters.

OPERATION_MODE
Can be **READ_WRITE** (default) or READ_ONLY.

CLEANUP_POLICY (STALE_QUERY_THRESHOLD_DAYS)
Configure the `STALE_QUERY_THRESHOLD_DAYS` argument to specify the number of days to retain data in the query store. The default value is 30. For SQL Database Basic edition, default is **7** days.

DATA_FLUSH_INTERVAL_SECONDS
Determines the frequency at which data written to the query store is persisted to disk. To optimize for performance, data collected by the query store is asynchronously written to the disk. The frequency at which this asynchronous transfer occurs is configured via `DATA_FLUSH_INTERVAL_SECONDS`. The default value is **900** (15 min).

MAX_STORAGE_SIZE_MB
Configures the maximum size of the query store. If the data in the query store hits the `MAX_STORAGE_SIZE_MB` limit, the query store automatically changes the state from read-write to read-only and stops collecting new data. The default value is 100 MB. For SQL Database Premium edition, default is **1 GB** and for SQL Database Basic edition, default is **10 MB**.

INTERVAL_LENGTH_MINUTES
Determines the time interval at which runtime execution statistics data is aggregated into the query store. To optimize for space usage, the runtime execution statistics in the Runtime Stats Store are aggregated over a fixed time window. This fixed time window is configured via `INTERVAL_LENGTH_MINUTES`. The default value is **60**.

SIZE_BASED_CLEANUP_MODE
Controls whether the cleanup process will be automatically activated when total amount of data gets close to maximum size. Can be **AUTO** (default) or OFF.

QUERY_CAPTURE_MODE
Designates if the Query Store captures all queries, or relevant queries based on execution count and resource consumption, or stops adding new queries and just tracks current queries. Can be **ALL** (capture all queries), AUTO (ignore infrequent and queries with insignificant compile and execution duration) or

NONE (stop capturing new queries). The default value on SQL Server (from SQL Server 2016 to SQL Server 2017) is ALL, while on Azure SQL Database is AUTO.

MAX_PLANS_PER_QUERY
An integer representing the maximum number of plans maintained for each query. The default value is **200**.

WAIT_STATS_CAPTURE_MODE
Controls if Query Store captures wait statistics information. Can be OFF or **ON** (default).

Query the **sys.database_query_store_options** view to determine the current options of the query store. For more information about the values, see sys.database_query_store_options.

For more information about setting options by using Transact-SQL statements, see Option Management.

Related Views, Functions, and Procedures

View and manage Query Storethrough Management Studio or by using the following views and procedures.

Query Store Functions

Functions help operations with the Query Store.

sys.fn_stmt_sql_handle_from_sql_stmt (Transact-SQL)

Query Store Catalog Views

Catalog views present information about the Query Store.

sys.database_query_store_options (Transact-SQL)	sys.query_context_settings (Transact-SQL)
sys.query_store_plan (Transact-SQL)	sys.query_store_query (Transact-SQL)
sys.query_store_query_text (Transact-SQL)	sys.query_store_runtime_stats (Transact-SQL)
sys.query_store_wait_stats (Transact-SQL)	sys.query_store_runtime_stats_interval (Transact-SQL)

Query Store Stored Procedures

Stored procedures configure the Query Store.