<https://learn.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server?view=sql-server-ver16>

# Cardinality Estimation (SQL Server)

* Article
* 01/08/2024
* 16 contributors

Feedback

## In this article

1. [Versions of the CE](https://learn.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server?view=sql-server-ver16#versions-of-the-ce)
2. [Use Query Store to assess the CE version](https://learn.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server?view=sql-server-ver16#use-query-store-to-assess-the-ce-version)
3. [Use extended events to assess the CE version](https://learn.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server?view=sql-server-ver16#use-extended-events-to-assess-the-ce-version)
4. [Steps to assess the CE version](https://learn.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server?view=sql-server-ver16#steps-to-assess-the-ce-version)

Show 4 more

**Applies to:**  [SQL Server](https://learn.microsoft.com/en-us/sql/sql-server/sql-docs-navigation-guide?view=sql-server-ver16#applies-to)  [Azure SQL Database](https://learn.microsoft.com/en-us/sql/sql-server/sql-docs-navigation-guide?view=sql-server-ver16#applies-to)  [Azure SQL Managed Instance](https://learn.microsoft.com/en-us/sql/sql-server/sql-docs-navigation-guide?view=sql-server-ver16#applies-to)

The SQL Server Query Optimizer is a cost-based Query Optimizer. This means that it selects query plans that have the lowest estimated processing cost to execute. The Query Optimizer determines the cost of executing a query plan based on two main factors:

* The total number of rows processed at each level of a query plan, referred to as the cardinality of the plan.
* The cost model of the algorithm dictated by the operators used in the query.

The first factor, cardinality, is used as an input parameter of the second factor, the cost model. Therefore, improved cardinality leads to better estimated costs and, in turn, faster execution plans.

Cardinality estimation (CE) in SQL Server is derived primarily from histograms that are created when indexes or statistics are created, either manually or automatically. Sometimes, SQL Server also uses constraint information and logical rewrites of queries to determine cardinality.

In the following cases, SQL Server can't accurately calculate cardinalities. This causes inaccurate cost calculations that might cause suboptimal query plans. Avoiding these constructs in queries might improve query performance. Sometimes, alternative query formulations or other measures are possible and these are pointed out:

* Queries with predicates that use comparison operators between different columns of the same table.
* Queries with predicates that use operators, and any one of the following are true:
  + There are no statistics on the columns involved on either side of the operators.
  + The distribution of values in the statistics isn't uniform, but the query seeks a highly selective value set. This situation can be especially true if the operator is anything other than the equality (=) operator.
  + The predicate uses the not equal to (!=) comparison operator or the NOT logical operator.
* Queries that use any of the SQL Server built-in functions or a scalar-valued, user-defined function whose argument isn't a constant value.
* Queries that involve joining columns through arithmetic or string concatenation operators.
* Queries that compare variables whose values aren't known when the query is compiled and optimized.

This article illustrates how you can assess and choose the best CE configuration for your system. Most systems benefit from the latest CE because it's the most accurate. The CE predicts how many rows your query will likely return. The cardinality prediction is used by the Query Optimizer to generate the optimal query plan. With more accurate estimations, the Query Optimizer can usually do a better job of producing a more optimal query plan.

Your application system could possibly have an important query whose plan is changed to a slower plan due to changes in the CE throughout versions. You have techniques and tools for identifying a query that performs slower due to CE issues. And you have options for how to address the ensuing performance issues.

## Versions of the CE

In 1998, a major update of the CE was part of SQL Server 7.0, for which the compatibility level was 70. This version of the CE model is set on four basic assumptions:

* **Independence:** Data distributions on different columns are assumed to be independent of each other, unless correlation information is available and usable.
* **Uniformity:** Distinct values are evenly spaced and that they all have the same frequency. More precisely, within each [histogram](https://learn.microsoft.com/en-us/sql/relational-databases/statistics/statistics?view=sql-server-ver16#histogram) step, distinct values are evenly spread and each value has same frequency.
* **Containment (Simple):** Users query for data that exists. For example, for an equality join between two tables, factor in the predicates selectivity1 in each input histogram, before joining histograms to estimate the join selectivity.
* **Inclusion:** For filter predicates where Column = Constant, the constant is assumed to actually exist for the associated column. If a corresponding histogram step is non-empty, one of the step's distinct values is assumed to match the value from the predicate.

1 Row count that satisfies the predicate.

Subsequent updates started with SQL Server 2014 (12.x), meaning compatibility levels 120 and above. The CE updates for levels 120 and above incorporate updated assumptions and algorithms that work well on modern data warehousing and on OLTP workloads. From the CE 70 assumptions, the following model assumptions were changed starting with CE 120:

* **Independence** becomes **Correlation:** The combination of the different column values is not necessarily independent. This might resemble more real-life data querying.
* **Simple Containment** becomes **Base Containment:** Users might query for data that does not exist. For example, for an equality join between two tables, we use the base tables histograms to estimate the join selectivity, and then factor in the predicates selectivity.

## Use Query Store to assess the CE version

Starting with SQL Server 2016 (13.x), the Query Store is a handy tool for examining the performance of your queries. Once Query Store is enabled, it will begin to track query performance over time, even if execution plans change. Monitor Query Store for high-cost or regressed query performance. For more information, see [Monitoring performance by using the Query Store](https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver16).

If preparing for an upgrade to SQL Server or promoting a database compatibility level in any SQL Server platform, consider [Upgrading Databases by using the Query Tuning Assistant](https://learn.microsoft.com/en-us/sql/relational-databases/performance/upgrade-dbcompat-using-qta?view=sql-server-ver16), which can help compare query performance in two different compatibility levels.

**Important**

Ensure the Query Store is correctly configured for your database and workload. For more information, see [**Best practices with Query Store**](https://learn.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store?view=sql-server-ver16).

## Use extended events to assess the CE version

Another option for tracking the cardinality estimation process is to use the extended event named query\_optimizer\_estimate\_cardinality. The following Transact-SQL code sample runs on SQL Server. It writes a .xel file to C:\Temp\ (although you can change the path). When you open the .xel file in Management Studio, its detailed information is displayed in a user friendly manner.

SQLCopy

DROP EVENT SESSION Test\_the\_CE\_qoec\_1 ON SERVER;

go

CREATE EVENT SESSION Test\_the\_CE\_qoec\_1

ON SERVER

ADD EVENT sqlserver.query\_optimizer\_estimate\_cardinality

(

ACTION (sqlserver.sql\_text)

WHERE (

sql\_text LIKE '%yourTable%'

and sql\_text LIKE '%SUM(%'

)

)

ADD TARGET package0.asynchronous\_file\_target

(SET

filename = 'c:\temp\xe\_qoec\_1.xel',

metadatafile = 'c:\temp\xe\_qoec\_1.xem'

);

GO

ALTER EVENT SESSION Test\_the\_CE\_qoec\_1

ON SERVER

STATE = START; --STOP;

GO

**Note**

The event sqlserver.query\_optimizer\_estimate\_cardinality is not available for Azure SQL Database.

For information about extended events as tailored for SQL Database, see [Extended events in SQL Database](https://learn.microsoft.com/en-us/azure/azure-sql/database/xevent-db-diff-from-svr).

## Steps to assess the CE version

Next are steps you can use to assess whether any of your most important queries perform worse under the latest CE. Some of the steps are performed by running a code sample presented in a preceding section.

1. Open SQL Server Management Studio (SSMS). Ensure your SQL Server database is set to the highest available compatibility level.
2. Perform the following preliminary steps:
   1. Open SQL Server Management Studio (SSMS).
   2. Run the Transact-SQL to ensure that your SQL Server database is set to the highest available compatibility level.
   3. Ensure that your database has its LEGACY\_CARDINALITY\_ESTIMATION configuration turned OFF.
   4. Clear your Query Store. In your database, ensure that [Query Store is ON](https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver16#Enabling).
   5. Run the statement: SET NOCOUNT OFF;
3. Run the statement: SET STATISTICS XML ON;
4. Run your important query.
5. In the results pane, on the **Messages** tab, note the actual number of rows affected.
6. In the results pane on the **Results** tab, double-click the cell that contains the statistics in XML format. A graphic query plan is displayed.
7. Right-click the first box in the graphic query plan, and then select **Properties**.
8. For later comparison with a different configuration, note the values for the following properties:
   1. **CardinalityEstimationModelVersion**.
   2. **Estimated Number of Rows**.
   3. **Estimated I/O Cost**, and several similar Estimated properties that involve actual performance rather than row count predictions.
   4. **Logical Operation** and **Physical Operation**. Parallelism is a good value.
   5. **Actual Execution Mode**. Batch is a good value, better than Row.
9. Compare the estimated number of rows to the actual number of rows. Is the CE inaccurate by 1% (high or low), or by 10%?
10. Run: SET STATISTICS XML OFF;
11. Run the Transact-SQL to decrease the compatibility level of your database by one level (such as from 130 down to 120).
12. Rerun all the non-preliminary steps.
13. Compare the CE property values from the two runs.
    1. Is the inaccuracy percentage under the newest CE less than under the older CE?
14. Finally, compare the various performance property values from the two runs.
    1. Did your query use a different plan under the two differing CE estimations?
    2. Did your query run slower under the latest CE?
    3. Unless your query runs better and with a different plan under the older CE, you almost certainly want the latest CE.
    4. However, if your query runs with a faster plan under the older CE, consider forcing the system to use the faster plan and to ignore the CE. This way you can have the latest CE on for everything, while keeping the faster plan in the one odd case.

## How to activate the best query plan

Suppose that with CE 120 or above, a less efficient query plan is generated for your query. Here are some options you have to activate the better plan, ordered from the largest scope to the smallest:

* You could set the database compatibility level to a value lower than the latest available, for your whole database.
  + For example, setting the compatibility level 110 or lower activates CE 70, but it makes all queries subject to the previous CE model.
  + Further, setting a lower compatibility level also misses a number of improvements in the query optimizer for latest versions, and affects all queries against the database.
* You could use LEGACY\_CARDINALITY\_ESTIMATION database scoped configuration option, to have the whole database use the older CE, while retaining other improvements in the query optimizer.
* You could use LEGACY\_CARDINALITY\_ESTIMATION query hint, to have a single query use the older CE, while retaining other improvements in the query optimizer.
* You could enforce the LEGACY\_CARDINALITY\_ESTIMATION via the Query Store hint feature, to have a single query use the older CE without changing the query.
* Force a different plan with Query Store.

### Database compatibility level

You can ensure your database is at a particular level by using the following Transact-SQL code for [COMPATIBILITY\_LEVEL](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql-compatibility-level?view=sql-server-ver16).

**Important**

The database engine version numbers for SQL Server and Azure SQL Database are not comparable with each other, and rather are internal build numbers for these separate products. The database engine for Azure SQL Server is based on the same code base as the SQL Server database engine. Most importantly, the database engine in Azure SQL Database always has the newest SQL database engine bits. Version 12 of Azure SQL Database is newer than version 15 of SQL Server. As of **November 2019**, in Azure SQL Database, the default compatibility level is 150 for newly created databases. Microsoft does not update Database Compatibility Level for existing databases. It is up to customers to do at their own discretion.

SQLCopy

SELECT ServerProperty('ProductVersion');

GO

SELECT d.name, d.compatibility\_level

FROM sys.databases AS d

WHERE d.name = 'yourDatabase';

GO

For pre-existing databases running at lower compatibility levels, as long as the application does not need to use enhancements that are only available in a higher database compatibility level, it is a valid approach to maintain the previous database compatibility level. For new development work, or when an existing application requires use of new features such as [Intelligent Query Processing](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?view=sql-server-ver16), as well as some new Transact-SQL, plan to upgrade the database compatibility level to the latest available. For more information, see [Compatibility levels and Database Engine upgrades](https://learn.microsoft.com/en-us/sql/database-engine/install-windows/compatibility-certification?view=sql-server-ver16#compatibility-levels-and-database-engine-upgrades).

**Caution**

Before changing database compatibility level, review [**Best Practices for upgrading Database Compatibility Level**](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql-compatibility-level?view=sql-server-ver16).

SQLCopy

ALTER DATABASE <yourDatabase>

SET COMPATIBILITY\_LEVEL = 150;

GO

For a SQL Server database set at compatibility level 120 or above, activation of the [trace flag 9481](https://learn.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-traceon-trace-flags-transact-sql?view=sql-server-ver16) forces the system to use the CE version 70.

### Legacy cardinality estimator

For a SQL Server database set at compatibility level 120 and above, the legacy cardinality estimator (CE version 70) can be activated at the database level by using the [ALTER DATABASE SCOPED CONFIGURATION](https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-database-scoped-configuration-transact-sql?view=sql-server-ver16).

SQLCopy

ALTER DATABASE SCOPED CONFIGURATION

SET LEGACY\_CARDINALITY\_ESTIMATION = ON;

GO

SELECT name, value

FROM sys.database\_scoped\_configurations

WHERE name = 'LEGACY\_CARDINALITY\_ESTIMATION';

GO

### Modify query to use hint

Starting with SQL Server 2016 (13.x) SP1, modify the query to use the [Query Hint](https://learn.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query?view=sql-server-ver16#use_hint) USE HINT ('FORCE\_LEGACY\_CARDINALITY\_ESTIMATION').

SQLCopy

SELECT CustomerId, OrderAddedDate

FROM OrderTable

WHERE OrderAddedDate >= '2016-05-01'

OPTION (USE HINT ('FORCE\_LEGACY\_CARDINALITY\_ESTIMATION'));

### Set a Query Store hint

Queries can be forced to use the legacy cardinality estimator without modifying the query, using [Query Store hints](https://learn.microsoft.com/en-us/sql/relational-databases/performance/query-store-hints?view=sql-server-ver16).

1. Identify the query in the [sys.query\_store\_query\_text](https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-query-store-query-text-transact-sql?view=sql-server-ver16) and [sys.query\_store\_query](https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-query-store-query-transact-sql?view=sql-server-ver16) Query Store catalog views. For example, search for an executed query by text fragment:

SQLCopy

SELECT q.query\_id, qt.query\_sql\_text

FROM sys.query\_store\_query\_text qt

INNER JOIN sys.query\_store\_query q ON

qt.query\_text\_id = q.query\_text\_id

WHERE query\_sql\_text like N'%ORDER BY ListingPrice DESC%'

AND query\_sql\_text not like N'%query\_store%';

1. The following example applies a Query Store hint to force the legacy cardinality estimator on query\_id 39, without modifying the query:

SQLCopy

EXEC sys.sp\_query\_store\_set\_hints @query\_id= 39, @query\_hints = N'OPTION(USE HINT(''FORCE\_LEGACY\_CARDINALITY\_ESTIMATION''))';

**Note**

For more information, see [**Query Story Hints**](https://learn.microsoft.com/en-us/sql/relational-databases/performance/query-store-hints?view=sql-server-ver16) (Preview). Currently, this feature is available only in Azure SQL Database.

### How to force a particular query plan

For the finest control, you could force the system to use the plan that was generated with CE 70 during your testing. After you pin your preferred plan, you can set your whole database to use the latest compatibility level and CE. The option is elaborated next.

The Query Store gives you different ways that you can force the system to use a particular query plan:

* Execute sys.sp\_query\_store\_force\_plan.
* In SQL Server Management Studio (SSMS), expand your **Query Store** node, right-click **Top Resource Consuming Nodes**, and then select **View Top Resource Consuming Nodes**. The display shows buttons labeled **Force Plan** and **Unforce Plan**.

For more information about the Query Store, see [Monitoring Performance By Using the Query Store](https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver16).

## Constant folding and expression evaluation during Cardinality Estimation

The Database Engine evaluates some constant expressions early to improve query performance. This is referred to as constant folding. A constant is a Transact-SQL literal, such as 3, 'ABC', '2005-12-31', 1.0e3, or 0x12345678. For more information, see [Constant Folding](https://learn.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver16#constant-folding-and-expression-evaluation).

In addition, some expressions that aren't constant folded but whose arguments are known at compile time, whether the arguments are parameters or constants, are evaluated by the result-set size (cardinality) estimator that is part of the Query Optimizer during optimization. For more information, see [Expression Evaluation](https://learn.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver16#constant-folding-and-expression-evaluation).

### Best Practices: Using constant folding and compile-time expression evaluation for generating optimal query plans

To make sure you generate optimal query plans, it's best to design queries, stored procedures, and batches so that the Query Optimizer can accurately estimate the selectivity of the conditions in your query, based on statistics about your data distribution. Otherwise, the Query Optimizer must use a default estimate when estimating selectivity.

To make sure that the Cardinality Estimator of the Query Optimizer provides good estimates, you should first make sure that the AUTO\_CREATE\_STATISTICS and AUTO\_UPDATE\_STATISTICS database SET options are ON (the default setting), or that you have manually created statistics on all columns referenced in a query condition. Then, when you're designing the conditions in your queries, do the following when it's possible:

* Avoid the use of local variables in queries. Instead, use parameters, literals, or expressions in the query.
* Limit the use of operators and functions embedded in a query that contains a parameter to those listed under Compile-Time Expression Evaluation for Cardinality Estimation.
* Make sure that constant-only expressions in the condition of your query are either constant-foldable, or can be evaluated at compilation time.
* If you have to use a local variable to evaluate an expression to be used in a query, consider evaluating it in a different scope than the query. For example, it might be helpful to perform one of the following options:
  + Pass the value of the variable to a stored procedure that contains the query you want to evaluate, and have the query use the procedure parameter instead of a local variable.
  + Construct a string that contains a query based in part on the value of the local variable, and then execute the string by using dynamic SQL (EXEC or preferably sp\_executesql).
  + Parameterize the query and execute it by using sp\_executesql, and pass the value of the variable as a parameter to the query.

## Examples of CE improvements

This section describes example queries that benefit from the enhancements implemented in the CE in recent releases. This is background information that doesn't call for specific action on your part.

### Example A. CE understands maximum value might be higher than when statistics were last gathered

Suppose statistics were last gathered for OrderTable on 2016-04-30, when the maximum OrderAddedDate was 2016-04-30. The CE 120 (and above version) understands that columns in OrderTable, which have ascending data might have values larger than the maximum recorded by the statistics. This understanding improves the query plan for Transact-SQL SELECT statements such as the following.

SQLCopy

SELECT CustomerId, OrderAddedDate

FROM OrderTable

WHERE OrderAddedDate >= '2016-05-01';

### Example B. CE understands that filtered predicates on the same table are often correlated

In the following SELECT we see filtered predicates on Model and ModelVariant. We intuitively understand that when Model is 'Xbox' there's a chance the ModelVariant is 'One', given that Xbox has a variant called One.

Starting with CE 120, SQL Server understands there might be a correlation between the two columns on the same table, Model and ModelVariant. The CE makes a more accurate estimation of how many rows will be returned by the query, and the [query optimizer](https://learn.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver16#optimizing-select-statements) generates a more optimal plan.

SQLCopy

SELECT Model, Purchase\_Price

FROM dbo.Hardware

WHERE Model = 'Xbox' AND

ModelVariant = 'Series X';

### Example C. CE no longer assumes any correlation between filtered predicates from different tables

Extensive new research on modern workloads and actual business data reveals that predicate filters from different tables usually don't correlate with each other. In the following query, the CE assumes there's no correlation between s.type and r.date. Therefore the CE makes a lower estimate of the number of rows returned.

SQLCopy

SELECT s.ticket, s.customer, r.store

FROM dbo.Sales AS s

CROSS JOIN dbo.Returns AS r

WHERE s.ticket = r.ticket AND

s.type = 'toy' AND

r.date = '2016-05-11';

## Related content

* [Cardinality estimation (CE) feedback](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing-cardinality-estimation-feedback?view=sql-server-ver16)
* [Monitor and Tune for Performance](https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitor-and-tune-for-performance?view=sql-server-ver16)
* [Optimizing Your Query Plans with the SQL Server 2014 Cardinality Estimator](https://learn.microsoft.com/en-us/previous-versions/dn673537(v=msdn.10))
* [Hints (Transact-SQL) - Query](https://learn.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query?view=sql-server-ver16)
* [USE HINT Query Hints](https://learn.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query?view=sql-server-ver16#use_hint)
* [Upgrade databases using the Query Tuning Assistant](https://learn.microsoft.com/en-us/sql/relational-databases/performance/upgrade-dbcompat-using-qta?view=sql-server-ver16)
* [Monitor performance by using the Query Store](https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver16)
* [Query processing architecture guide](https://learn.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver16)
* [Query Store hints](https://learn.microsoft.com/en-us/sql/relational-databases/performance/query-store-hints?view=sql-server-ver16)
* [Intelligent query processing in SQL databases](https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?view=sql-server-ver16)
* [DBCC TRACEON - Trace Flags (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-traceon-trace-flags-transact-sql?view=sql-server-ver16)
* [sys.query\_store\_query\_hints](https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-query-store-query-hints-transact-sql?view=sql-server-ver16)