Statistics

# Statistics

The Query Optimizer uses statistics to create query plans that improve query performance. For most queries, the Query Optimizer already generates the necessary statistics for a high quality query plan; in some cases, you need to create additional statistics or modify the query design for best results.

Statistics for query optimization are binary large objects (BLOBs) that contain statistical information about the distribution of values in one or more columns of a table or indexed view. The Query Optimizer uses these statistics to estimate the cardinality, or number of rows, in the query result. These cardinality estimates enable the Query Optimizer to create a high-quality query plan. For example, depending on your predicates, the Query Optimizer could use cardinality estimates to choose the index seek operator instead of the more resource-intensive index scan operator, and in doing so improve query performance.

Each statistics object is created on a list of one or more table columns and includes a histogram displaying the distribution of values in the first column. Statistics objects on multiple columns also store statistical information about the correlation of values among the columns. These correlation statistics, or densities, are derived from the number of distinct rows of column values.

## Histogram

A **histogram** measures the frequency of occurrence for each distinct value in a data set. The query optimizer computes a histogram on the column values in the first key column of the statistics object, selecting the column values by statistically sampling the rows or by performing a full scan of all rows in the table or view. If the histogram is created from a sampled set of rows, the stored totals for number of rows and number of distinct values are estimates and do not need to be whole integers.

Note

Histograms in SQL Server are only built for a single column—the first column in the set of key columns of the statistics object.

## When to create statistics

The Query Optimizer already creates statistics in the following ways:

1. The Query Optimizer creates statistics for indexes on tables or views when the index is created. These statistics are created on the key columns of the index. If the index is a filtered index, the Query Optimizer creates filtered statistics on the same subset of rows specified for the filtered index. For more information about filtered indexes,
2. The Query Optimizer creates statistics for single columns in query predicates when [AUTO\_CREATE\_STATISTICS](https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql-set-options#auto_create_statistics) is on.

Consider updating statistics for the following conditions:

* Query execution times are slow.
* Insert operations occur on ascending or descending key columns.
* After maintenance operations.

## Queries that use statistics effectively

### Improving cardinality estimates for expressions

WHERE PRICE + Tax > 100

### Improving cardinality estimates for variables and functions

* If the query predicate uses a local variable, consider rewriting the query to use a parameter instead of a local variable. The value of a local variable is not known when the Query Optimizer creates the query execution plan. When a query uses a parameter, the Query Optimizer uses the cardinality estimate for the first actual parameter value that is passed to the stored procedure.
* Consider using a standard table or temporary table to hold the results of multi-statement table-valued functions (mstvf). The Query Optimizer does not create statistics for multi-statement table-valued functions. With this approach the Query Optimizer can create statistics on the table columns and use them to create a better query plan.
* Consider using a standard table or temporary table as a replacement for table variables. The Query Optimizer does not create statistics for table variables. With this approach the Query Optimizer can create statistics on the table columns and use them to create a better query plan. There are tradeoffs in determining whether to use a temporary table or a table variable; Table variables used in stored procedures cause fewer recompilations of the stored procedure than temporary tables. Depending on the application, using a temporary table instead of a table variable might not improve performance.
* If a stored procedure contains a query that uses a passed-in parameter, avoid changing the parameter value within the stored procedure before using it in the query. The cardinality estimates for the query are based on the passed-in parameter value and not the updated value. To avoid changing the parameter value, you can rewrite the query to use two stored procedures.

### Improving cardinality estimates with query hints

To improve cardinality estimates for local variables, you can use the OPTIMIZE FOR <value> or OPTIMIZE FOR UNKNOWN query hints with RECOMPILE.

### Improving cardinality estimates with Plan Guides

## Create Statistics

CREATE STATISTICS ContactMail1

ON Person.Person (BusinessEntityID, EmailPromotion);

## DROP STATISTICS

DROP STATISTICS table.statistics\_name | view.statistics\_name [ ,...n ]

DBCC SHOW\_STATISTICS ("Person.Address", AK\_Address\_rowguid);

## View Statistics Properties

SELECT name AS statistics\_name ,stats\_id ,auto\_created ,user\_created ,no\_recompute ,has\_filter ,filter\_definition

FROM sys.stats -- for the Sales.SpecialOffer table

WHERE object\_id = OBJECT\_ID('Sales.SpecialOffer');

## Rename Statistics

EXEC sp\_rename N'AK\_Employee\_LoginID', N'AK\_Emp\_Login', N'STATISTICS';

## Update Statistics

UPDATE STATISTICS Sales.SalesOrderDetail AK\_SalesOrderDetail\_rowguid;

# Stored Procedures

A stored procedure in SQL Server is a group of one or more Transact-SQL statements or a reference to a Microsoft .NET Framework common runtime language (CLR) method.

* Accept input parameters and return multiple values in the form of output parameters to the calling program.
* Contain programming statements that perform operations in the database. These include calling other procedures.
* Return a status value to a calling program to indicate success or failure (and the reason for failure).

## Benefits of Using Stored Procedures

**Reduced server/client network traffic**  
The commands in a procedure are executed as a single batch of code. This can significantly reduce network traffic between the server and client because only the call to execute the procedure is sent across the network.

**Stronger security**  
Multiple users and client programs can perform operations on underlying database objects through a procedure, even if the users and programs do not have direct permissions on those underlying objects.

The [EXECUTE AS](https://docs.microsoft.com/en-us/sql/t-sql/statements/execute-as-clause-transact-sql) clause can be specified in the CREATE PROCEDURE statement to enable impersonating another user, or enable users or applications to perform certain database activities without needing direct permissions on the underlying objects and commands. For example, some actions such as TRUNCATE TABLE, do not have grantable permissions. To execute TRUNCATE TABLE, the user must have ALTER permissions on the specified table. Granting a user ALTER permissions on a table may not be ideal because the user will effectively have permissions well beyond the ability to truncate a table. By incorporating the TRUNCATE TABLE statement in a module and specifying that module execute as a user who has permissions to modify the table, you can extend the permissions to truncate the table to the user that you grant EXECUTE permissions on the module.

When calling a procedure over the network, only the call to execute the procedure is visible. Therefore, malicious users cannot see table and database object names, embed Transact-SQL statements of their own, or search for critical data.

Procedures can be encrypted, helping to obfuscate the source code

**Reuse of code**The code for any repetitious database operation is the perfect candidate for encapsulation in procedures. This eliminates needless rewrites of the same code, decreases code inconsistency, and allows the code to be accessed and executed by any user or application possessing the necessary permissions.

**Easier maintenance**  
When client applications call procedures and keep database operations in the data tier, only the procedures must be updated for any changes in the underlying database. The application tier remains separate and does not have to know how about any changes to database layouts, relationships, or processes.

**Improved performance**  
By default, a procedure compiles the first time it is executed and creates an execution plan that is reused for subsequent executions. Since the query processor does not have to create a new plan, it typically takes less time to process the procedure.

## Types of Stored Procedures

**User-defined**  
A user-defined procedure can be created in a user-defined database or in all system databases except the **Resource** database. The procedure can be developed in either Transact-SQL or as a reference to a Microsoft .NET Framework common runtime language (CLR) method.

**Temporary**  
Temporary procedures are a form of user-defined procedures. The temporary procedures are like a permanent procedure, except temporary procedures are stored in **tempdb**. There are two types of temporary procedures: local and global. They differ from each other in their names, their visibility, and their availability. Local temporary procedures have a single number sign (#) as the first character of their names; they are visible only to the current user connection, and they are deleted when the connection is closed. Global temporary procedures have two number signs (##) as the first two characters of their names; they are visible to any user after they are created, and they are deleted at the end of the last session using the procedure.

**System**  
System procedures are included with SQL Server. They are physically stored in the internal, hidden **Resource** database and logically appear in the **sys** schema of every system- and user-defined database. In addition, the **msdb** database also contains system stored procedures in the **dbo** schema that are used for scheduling alerts and jobs. Because system procedures start with the prefix **sp\_**, we recommend that you do not use this prefix when naming user-defined procedures.

**Extended User-Defined**Extended procedures enable creating external routines in a programming language such as C. These procedures are DLLs that an instance of SQL Server can dynamically load and run.

Note

Extended stored procedures will be removed in a future version of SQL Server. Do not use this feature in new development work, and modify applications that currently use this feature as soon as possible. Create CLR procedures instead. This method provides a more robust and secure alternative to writing extended procedures.