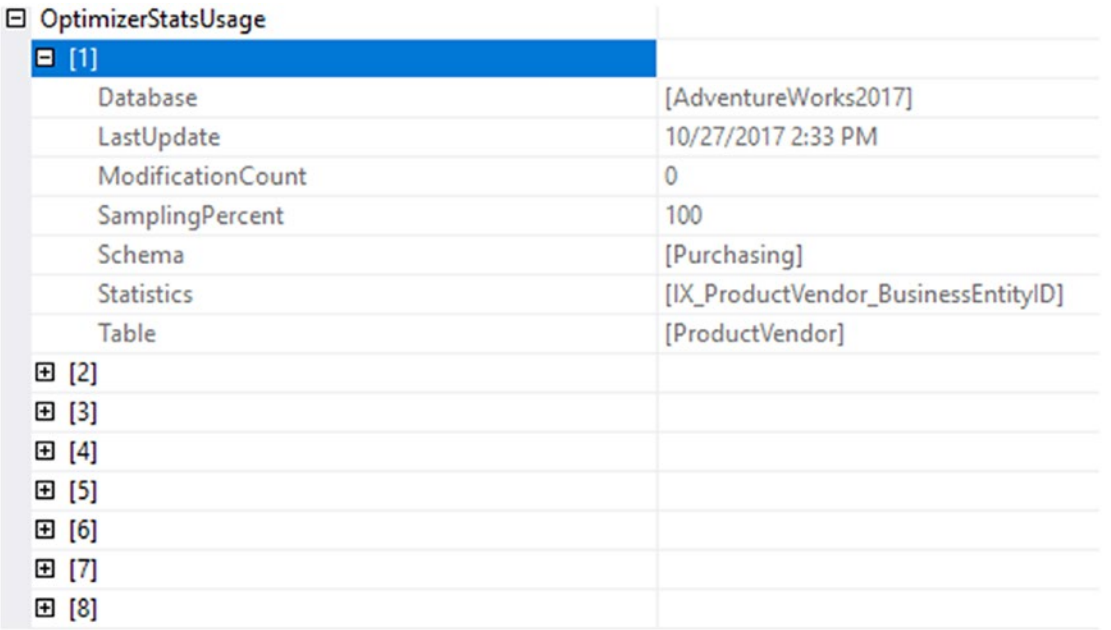


Figure 13-30. *Statistics in use within the execution plan generated for a query*

Enabling and Disabling the Cardinality Estimator

If you create a database in SQL Server 2014 or greater, it's going to automatically come with the compatibility level set to 120, or greater, which is the correct version for the latest SQL Server. But, if you restore or attach a database from a previous version of SQL Server, the compatibility level will be set to that version, 110 or before. That database will then use the SQL Server 7 cardinality estimator. You can tell this by looking at the execution plan in the first operator (`SELECT/INSERT/UPDATE/DELETE`) at the properties for the `CardinalityEstimationModelVersion`, as shown in Figure 13-31.

Figure 13-31. *Property in the first operator showing the cardinality estimator in use*

The value shown for SQL Server 2014–2017 will correspond to the version, 120, 130, 140. That's how you can tell what version of the cardinality estimator is in use. This is important because since the estimates can lead to changes in execution plans, it's really

important that you understand how to troubleshoot the issues in the event that you get a degradation in performance caused by the new cardinality estimations.

If you suspect that you are experiencing problems from the upgrade, you should absolutely compare your actual rows returned to the estimated rows returned in the operations within the execution plan. That's always a great way to determine whether statistics or cardinality estimations are causing you issues. You should be using the Query Store for both testing your upgrades and as part of the upgrade process (as outlined in Chapter 11). The Query Store is the best way to capture before and after the change in the cardinality estimation engine and the best way to deal with the individual queries that may go wrong.

You have the option of disabling the new cardinality estimation functionality by setting the compatibility level to 110, but that also disables other newer SQL Server functionality, so it might not be a good choice. You can run a trace flag against the restore of the database using `OPTION (QUERYTRACEON 9481)`; you'll target just the cardinality estimator for that database. If you determine in a given query that you're having issues with the new cardinality estimator, you can take advantage of trace flags in the query in the same way.

```
SELECT p.Name,
       p.Class
FROM Production.Product AS p
WHERE p.Color = 'Red'
      AND p.DaysToManufacture > 15
OPTION (QUERYTRACEON 9481);
```

Conversely, if you have turned off the cardinality estimator using the trace flag or compatibility level, you can selectively turn it on for a given query using the same functionality as earlier but substituting 2312 for the trace flag value.

Finally, a new function was introduced in SQL Server 2016, Database Scoped Configuration. Among other settings (which we'll discuss in appropriate places throughout the book), you can disable just the cardinality estimation engine without disabling all the modern functionality. The new syntax looks like this:

```
ALTER DATABASE SCOPED CONFIGURATION SET LEGACY_CARDINALITY_ESTIMATION = ON;
```

Using this command, you can change the behavior of the database without changing all other behaviors. You can also use the same command to turn off the legacy cardinality estimator. You also have the option of a USE hint on individual queries. Setting FORCE_LEGACY_CARDINALITY_ESTIMATION inside a query hint will make that query use the old cardinality estimation, and only that one query. This is probably the single safest option, although it does involve code changes.

Statistics DMOs

Prior to SQL Server 2016, the only way to get information on statistics was to use DBCC SHOW_STATISTICS. However, a couple of new DMFs have been introduced that can be useful. The sys.dm_db_stats_properties function returns the header information of a set of statistics. This means you quickly pull information out of the header. For example, use this query to retrieve when the statistics were last updated:

```
SELECT ddsp.object_id,
       ddsp.stats_id,
       ddsp.last_updated
FROM sys.dm_db_stats_properties(OBJECT_ID('HumanResources.Employee'),
                                2) AS ddsp;
```

The function requires that you pass the object_id that you're interested in and the statistics_id for that object. In this example we look at the column statistics on the HumanResources.Employee table.

The other function is sys.dm_db_stats_histogram. It works much the same way, allowing us to treat the histogram of statistics as a queryable object. For example, suppose we wanted to find a particular set of values within the histogram. Normally, you look for the range_hi_key value and then see whether the value you're looking for is less than one range_high_key but greater than another. It's entirely possible to automate this now.

```
WITH histo
AS (SELECT ddsh.step_number,
           ddsh.range_high_key,
           ddsh.range_rows,
           ddsh.equal_rows,
           ddsh.average_range_rows
```

```
    FROM sys.dm_db_stats_histogram(OBJECT_ID('HumanResources.Employee'),
                                   1) AS ddsh ),
     histojoin
AS (SELECT h1.step_number,
           h1.range_high_key,
           h2.range_high_key AS range_high_key_step1,
           h1.range_rows,
           h1.equal_rows,
           h1.average_range_rows
    FROM histo AS h1
        LEFT JOIN histo AS h2
            ON h1.step_number = h2.step_number + 1)
SELECT hj.range_high_key,
       hj.equal_rows,
       hj.average_range_rows
FROM histojoin AS hj
WHERE hj.range_high_key >= 17
      AND (   hj.range_high_key_step1 < 17
              OR hj.range_high_key_step1 IS NULL);
```

This query will look through the statistics in question on the `HumanResources.Employee` table and will find which row in the histogram would contain the value of 17.

Statistics Maintenance

SQL Server allows a user to manually override the maintenance of statistics in an individual database. The four main configurations controlling the automatic statistics maintenance behavior of SQL Server are as follows:

- New statistics on columns with no index (auto create statistics)
- Updating existing statistics (auto update statistics)
- The degree of sampling used to generate statistics
- Asynchronous updating of existing statistics (auto update statistics async)

You can control the preceding configurations at the levels of a database (all indexes and statistics on all tables) or on a case-by-case basis on individual indexes or statistics. The auto create statistics setting is applicable for nonindexed columns only because SQL Server always creates statistics for an index key when the index is created. The auto update statistics setting, and the asynchronous version, is applicable for statistics on both indexes and statistics on columns with no index.

Automatic Maintenance

By default, SQL Server automatically takes care of statistics. Both the auto create statistics and auto update statistics settings are on by default. As explained previously, it is usually better to keep these settings on. The auto update statistics async setting is off by default.

When you rebuild an index (if you choose to rebuild an index), it will create all new statistics for that index, based on a full scan of the data (more on that coming up). This means the rebuild process results in a very high-quality set of statistics, yet another way Microsoft helps you maintain your statistics automatically.

However, situations arise where creating and maintaining your statistics manually works better. For many of us, ensuring that our statistics are more up-to-date than the automated processes makes them means a higher degree of workload predictability. We know when and how we're maintaining the statistics because they are under our control. You also get to stop statistics maintenance from occurring randomly and control exactly when they occur, as well as control the recompiles that they lead to. This helps focus the load on your production system to nonpeak hours.

Auto Create Statistics

The auto create statistics feature automatically creates statistics on nonindexed columns when referred to in the `WHERE` clause of a query. For example, when this `SELECT` statement is run against the `Sales.SalesOrderHeader` table on a column with no index, statistics for the column are created:

```
SELECT  cc.CardNumber,
        cc.ExpMonth,
        cc.ExpYear
FROM    Sales.CreditCard AS cc
WHERE   cc.CardType = 'Vista';
```

Then the auto create statistics feature (make sure it is turned back on if you have turned it off) automatically creates statistics on column `CardType`. You can see this in the Extended Events session output in Figure 13-32.

name	batch_text	job_type	statistics_list
auto_stats	NULL	StatsUpdate	Created: CardType
auto_stats	NULL	StatsUpdate	Loading without updating: Sales.CreditCard._WA_Sys_000000...
auto_stats	NULL	StatsUpdate	Loading without updating: Sales.CreditCard.AK_CreditCard_Car...
sql_batch_completed	SELECT cc.CardNumber, cc.Ex...	NULL	NULL

Figure 13-32. *Session output with AUTO_CREATE_STATISTICS ON*

The `auto_stats` event fires to create the new set of statistics. You can see the details of what is happening in the `statistics_list` field `Created: CardType`. This is followed by the loading process of the new column statistic and a statistic on one of the indexes on the table and, finally, by the execution of the query.

Auto Update Statistics

The auto update statistics feature automatically updates existing statistics on the indexes and columns of a permanent table when the table is referred to in a query, provided the statistics have been marked as out-of-date. The types of changes are action statements, such as `INSERT`, `UPDATE`, and `DELETE`. The default threshold for the number of changes depends on the number of rows in the table. It's a fairly simple calculation.

```
Sqrt(1000*NumberOfRows)
```

This means if you had 500,000 rows in a table, then plugging that into the calculation results in 22,360.68. You would need to add, edit, or delete that many rows in your 500,000-row table before an automatic statistics update would occur.

For SQL Server 2014 and earlier, when not running under trace flag 2371, statistics are maintained as shown in Table 13-4.

Table 13-4. *Update Statistics Threshold for Number of Changes*

Number of Rows	Threshold for Number of Changes
0	> 1 insert
<500	> 500 changes
>500	20 percent of row changes

Row changes are counted as the number of inserts, updates, or deletes in the table.

Using a threshold reduces the frequency of the automatic update of statistics. For example, consider the following table:

```
IF (SELECT OBJECT_ID('dbo.Test1')) IS NOT NULL
    DROP TABLE dbo.Test1;

CREATE TABLE dbo.Test1 (C1 INT);

CREATE INDEX ixl ON dbo.Test1 (C1);

INSERT INTO dbo.Test1 (C1)
VALUES (0);
```

After the nonclustered index is created, a single row is added to the table. This outdates the existing statistics on the nonclustered index. If the following `SELECT` statement is executed with a reference to the indexed column in the `WHERE` clause, like so, then the auto update statistics feature automatically updates statistics on the nonclustered index, as shown in the session output in Figure 13-33:

```
SELECT  C1
FROM    dbo.Test1
WHERE   C1 = 0;
```

Field	Value
async	False
attach_activity_id.g...	801C11DA-4063-4F21-A095-10655B72BB3A
attach_activity_id.s...	3
count	1
database_id	6
database_name	
duration	0
incremental	False
index_id	2
job_id	0
job_type	StatsUpdate
last_error	0
max_dop	-1
object_id	724197630
retries	0
sample_percentage	-1
statistics_list	Loading and updating: dbo.Test1.ix
status	Loading and updating stats
success	True

Figure 13-33. *Session output with AUTO_UPDATE_STATISTICS ON*

Once the statistics are updated, the change-tracking mechanisms for the corresponding tables are set to 0. This way, SQL Server keeps track of the number of changes to the tables and manages the frequency of automatic updates of statistics.

The new functionality of SQL Server 2016 and newer means that for larger tables, you will get more frequent statistics updates. You'll need to take advantage of trace flag 2371 on older versions of SQL Server to arrive at the same functionality. If automatic updates are not occurring frequently enough, you can take direct control, discussed in the "Manual Maintenance" section later in this chapter.

Auto Update Statistics Asynchronously

If auto update statistics asynchronously is set to on, the basic behavior of statistics in SQL Server isn't changed radically. When a set of statistics is marked as out-of-date and a query is then run against those statistics, the statistics update does not interrupt the execution of the query, like normally happens. Instead, the query finishes execution using the older set of statistics. Once the query completes, the statistics are updated. The reason this may be attractive is that when statistics are updated, query plans in the procedure cache are removed, and the query being run must be recompiled. This causes a delay in the execution of the query. So, rather than make a query wait for both the update of the statistics and a recompile of the procedure, the query completes its run. The next time the same query is called, it will have updated statistics waiting for it, and it will have to recompile only.

Although this functionality does make the steps needed to update statistics and recompile the procedure somewhat faster, it can also cause queries that could benefit immediately from updated statistics and a new execution plan to work with the old execution plan. Careful testing is required before turning this functionality on to ensure it doesn't cause more harm than good.

Note If you are attempting to update statistics asynchronously, you must also have `AUTO_UPDATE_STATISTICS` set to `ON`.

Manual Maintenance

The following are situations in which you need to interfere with or assist the automatic maintenance of statistics:

- *When experimenting with statistics*: Just a friendly suggestion—please spare your production servers from experiments such as the ones you are doing in this book.
- *After upgrading from a previous version to SQL Server 2017*: In earlier versions of this book I suggested updating statistics immediately on an upgrade to a new version of SQL Server. This was because of the changes in statistics introduced in SQL Server 2014. It made sense to immediately update the statistics so that you were seeing the effects

of the new cardinality estimator. With the addition of the Query Store, I can no longer make this recommendation in the same way. Instead, I'll suggest that you consider it if upgrading from SQL Server 2014 to a newer version, but even then, I wouldn't suggest manually updating the statistics by default. I would test it first to understand how a given upgrade will behave.

- *While executing a series of ad hoc SQL activities that you won't execute again*: In such cases, you must decide whether you want to pay the cost of automatic statistics maintenance to get a better plan for that one case while affecting the performance of other SQL Server activities. So, in general, you might not need to be concerned with such singular events. This is mainly applicable to larger databases, but you can test it in your environment if you think it may apply.
- *When you come upon an issue with the automatic statistics maintenance and the only workaround for the time being is to keep the automatic statistics maintenance feature off*: Even in these cases, you can turn the feature off for the specific table that faces the problem instead of disabling it for the complete database. Issues like this can be found in large data sets where the data is updated a lot but not enough to trigger the threshold update. Also, it can be used in cases where the sampling level of the automatic updates is not adequate for some data distributions.
- *While analyzing the performance of a query, you realize that the statistics are missing for a few of the database objects referred to by the query*: This can be evaluated from the graphical and XML execution plans, as explained earlier in the chapter.
- *While analyzing the effectiveness of statistics, you realize that they are inaccurate*: This can be determined when poor execution plans are being created from what should be good sets of statistics.

SQL Server allows a user to control many of its automatic statistics maintenance features. You can enable (or disable) the automatic statistics creation and update features by using the auto create statistics and auto update statistics settings, respectively, and then you can get your hands dirty.

Manage Statistics Settings

You can control the auto create statistics setting at a database level. To disable this setting, use the `ALTER DATABASE` command.

```
ALTER DATABASE AdventureWorks2017 SET AUTO_CREATE_STATISTICS OFF;
```

You can control the auto update statistics setting at different levels of a database, including all indexes and statistics on a table, or at the individual index or statistics level. To disable auto update statistics at the database level, use the `ALTER DATABASE` command.

```
ALTER DATABASE AdventureWorks2017 SET AUTO_UPDATE_STATISTICS OFF;
```

Disabling this setting at the database level overrides individual settings at lower levels. Auto update statistics asynchronously requires that the auto update statistics be on first. Then you can enable the asynchronous update.

```
ALTER DATABASE AdventureWorks2017 SET AUTO_UPDATE_STATISTICS_ASYNC ON;
```

To configure auto update statistics for all indexes and statistics on a table in the current database, use the `sp_autostats` system stored procedure.

```
USE AdventureWorks2017;
EXEC sp_autostats
    'HumanResources.Department',
    'OFF';
```

You can also use the same stored procedure to configure this setting for individual indexes or statistics. To disable this setting for the `AK_Department_Name` index on `AdventureWorks2017.HumanResources.Department`, execute the following statements:

```
EXEC sp_autostats
    'HumanResources.Department',
    'OFF',
    AK_Department_Name;
```

You can also use the `UPDATE STATISTICS` command's `WITH NORECOMPUTE` option to disable this setting for all or individual indexes and statistics on a table in the current database. The `sp_createstats` stored procedure also has the `NORECOMPUTE` option. The `NORECOMPUTE` option will not disable automatic update of statistics for the database, but it will for a given set of statistics.

Avoid disabling the automatic statistics features, unless you have confirmed through testing that this brings a performance benefit. If the automatic statistics features are disabled, then you are responsible for manually identifying and creating missing statistics on the columns that are not indexed and then keeping the existing statistics up-to-date. In general, you're only going to want to disable the automatic statistics features for very large tables and only after you've carefully measured the blocking and locking so that you know that changing statistics behavior will help.

If you want to check the status of whether a table has its automatic statistics turned off, you can use this:

```
EXEC sp_autostats 'HumanResources.Department';
```

Reset the automatic maintenance of the index so that it is on where it has been turned off.

```
EXEC sp_autostats
    'HumanResources.Department',
    'ON';
EXEC sp_autostats
    'HumanResources.Department',
    'ON',
    AK_Department_Name;
```

Generate Statistics

To create statistics manually, use one of the following options:

- `CREATE STATISTICS`: You can use this option to create statistics on single or multiple columns of a table or an indexed view. Unlike the `CREATE INDEX` command, `CREATE STATISTICS` uses sampling by default.
- `sys.sp_createstats`: Use this stored procedure to create single-column statistics for all eligible columns for all user tables in the current database. This includes all columns except computed columns; columns with the `NTEXT`, `TEXT`, `GEOMETRY`, `GEOGRAPHY`, or `IMAGE` data type; sparse columns; and columns that already have statistics or are the first column of an index. This function is meant for backward compatibility, and I don't recommend using it.

While a statistics object is created for a columnstore index, the values inside that index are null. Individual columns on a columnstore index can have the regular system-generated statistics created against them. When dealing with a columnstore index, if you find that you're still referencing the individual columns, you may find, in some situations, that creating a multicolumn statistic is useful. An example would look like this:

```
CREATE STATISTICS MultiColumnExample
ON dbo.bigProduct (ProductNumber,
                   Name);
```

With the exception of the individual column statistics and any that you create, there is no need to worry about the automatically created index statistic on a columnstore index.

If you partition a columnstore index (partitioning is not a performance enhancement tool, it's a data management tool), you'll need to change your statistics to be incremental using the following command to ensure that statistics updates are only by partition:

```
UPDATE STATISTICS dbo.bigProduct WITH RESAMPLE, INCREMENTAL=ON;
```

To update statistics manually, use one of the following options:

- `UPDATE STATISTICS`: You can use this option to update the statistics of individual or all index keys and nonindexed columns of a table or an indexed view.
- `sys.sp_updatestats`: Use this stored procedure to update statistics of all user tables in the current database. However, note that it can only sample the statistics, not use `FULLSCAN`, and it will update statistics when only a single action has been performed on that statistics. In short, this is a rather blunt instrument for maintaining statistics.

You may find that allowing the automatic updating of statistics is not quite adequate for your system. Scheduling `UPDATE STATISTICS` for the database during off-hours is an acceptable way to deal with this issue. `UPDATE STATISTICS` is the preferred mechanism because it offers a greater degree of flexibility and control. It's possible, because of the types of data inserted, that the sampling method for gathering the statistics, used because it's faster, may not gather the appropriate data. In these cases, you can force a `FULLSCAN` so that all the data is used to update the statistics just like what happens when the statistics are initially created. This can be a costly operation, so it's best to be

selective about which indexes receive this treatment and when it is run. In addition, if you do set sampling rates for your statistics rebuilds, including FULLSCAN, you should use PERSIST_SAMPLE_PERCENT to ensure that any automated processes that fire will use the same sampling rate.

Note In general, you should always use the default settings for automatic statistics. Consider modifying these settings only after identifying that the default settings appear to detract from performance.

Statistics Maintenance Status

You can verify the current settings for the autostats feature using the following:

- sys.databases
- DATABASEPROPERTYEX
- sp_autostats

Status of Auto Create Statistics

You can verify the current setting for auto create statistics by running a query against the sys.databases system table.

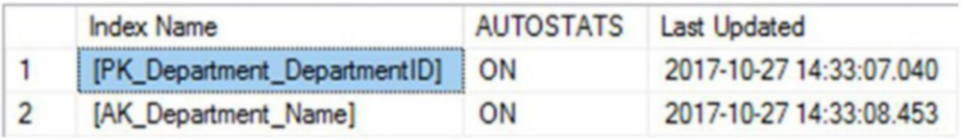
```
SELECT  is_auto_create_stats_on
FROM    sys.databases
WHERE   [name] = 'AdventureWorks2017';
```

A return value of 1 means enabled, and a value of 0 means disabled.

You can also verify the status of specific indexes using the sp_autostats system stored procedure, as shown in the following code. Supplying any table name to the stored procedure will provide the configuration value of auto create statistics for the current database under the Output section of the global statistics settings.

```
USE AdventureWorks2017;
EXEC sys.sp_autostats 'HumanResources.Department';
```

Figure 13-34 shows an excerpt of the preceding `sp_autostats` statement's output.



	Index Name	AUTOSTATS	Last Updated
1	[PK_Department_DepartmentID]	ON	2017-10-27 14:33:07.040
2	[AK_Department_Name]	ON	2017-10-27 14:33:08.453

Figure 13-34. *`sp_autostats` output*

A return value of `ON` means enabled, and a value of `OFF` means disabled. This stored procedure is more useful when verifying the status of auto update statistics, as explained earlier in this chapter.

You can also verify the current setting for auto update statistics, and auto update statistics asynchronously, in a similar manner to auto create statistics. Here's how to do it using the function `DATABASEPROPERTYEX`:

```
SELECT  DATABASEPROPERTYEX('AdventureWorks2017', 'IsAutoUpdateStatistics');
```

Here's how to do it using `sp_autostats`:

```
USE AdventureWorks2017;
EXEC sp_autostats
    'Sales.SalesOrderDetail';
```

Analyzing the Effectiveness of Statistics for a Query

For performance reasons, it is extremely important to maintain proper statistics on your database objects. Issues with statistics can be fairly common. You need to keep your eyes open to the possibility of problems with statistics while analyzing the performance of a query. If an issue with statistics does arise, then it can really take you for a ride. In fact, checking that the statistics are up-to-date at the beginning of a query-tuning session eliminates an easily fixed problem. In this section, you'll see what you can do should you find statistics to be missing or out-of-date.

While analyzing an execution plan for a query, look for the following points to ensure a cost-effective processing strategy:

- Indexes are available on the columns referred to in the filter and join criteria.
- In the case of a missing index, statistics should be available on the columns with no index. It may be preferable to have the index itself.
- Since outdated statistics are of no use and can even be misleading, it is important that the estimates used by the optimizer from the statistics are up-to-date.

You analyzed the use of a proper index in Chapter 9. In this section, you will analyze the effectiveness of statistics for a query.

Resolving a Missing Statistics Issue

To see how to identify and resolve a missing statistics issue, consider the following example. To more directly control the data, I'll use a test table instead of one of the AdventureWorks2017 tables. First disable both auto create statistics and auto update statistics using the `ALTER DATABASE` command.

```
ALTER DATABASE AdventureWorks2017 SET AUTO_CREATE_STATISTICS OFF;
ALTER DATABASE AdventureWorks2017 SET AUTO_UPDATE_STATISTICS OFF;
```

Create a test table with a large number of rows and a nonclustered index.

```
IF EXISTS (   SELECT *
              FROM sys.objects
              WHERE object_id = OBJECT_ID(N'dbo.Test1'))
    DROP TABLE dbo.Test1;
GO

CREATE TABLE dbo.Test1 (C1 INT,
                        C2 INT,
                        C3 CHAR(50));
INSERT INTO dbo.Test1 (C1,
                       C2,
                       C3)
```

```
VALUES (51, 1, 'C3'),
       (52, 1, 'C3');

CREATE NONCLUSTERED INDEX iFirstIndex ON dbo.Test1 (C1, C2);

SELECT TOP 10000
       IDENTITY(INT, 1, 1) AS n
INTO #Nums
FROM master.dbo.syscolumns AS scl,
     master.dbo.syscolumns AS sC2;

INSERT INTO dbo.Test1 (C1,
                       C2,
                       C3)
SELECT n % 50,
       n,
       'C3'
FROM #Nums;
DROP TABLE #Nums;
```

Since the index is created on (C1, C2), the statistics on the index contain a histogram for the first column, C1, and density values for the prefixed column combinations (C1 and C1 * C2). There are no histograms or density values alone for column C2.

To understand how to identify missing statistics on a column with no index, execute the following SELECT statement. Since the auto create statistics feature is off, the optimizer won't be able to find the data distribution for the column C2 used in the WHERE clause. Before executing the query, ensure you have enabled Include Actual Execution Plan by clicking the query toolbar or hitting Ctrl+M.

```
SELECT  *
FROM    dbo.Test1
WHERE   C2 = 1;
```

The information on missing statistics is also provided by the graphical execution plan, as shown in Figure 13-35.

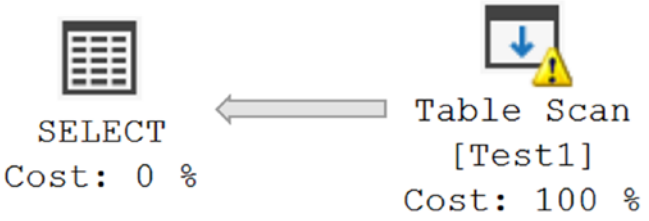


Figure 13-35. *Missing statistics indication in a graphical plan*

The graphical execution plan shows an operator with the yellow exclamation point. This indicates some problem with the operator in question. You can obtain a detailed description of the warning by right-clicking the Table Scan operator and then selecting Properties from the context menu. There's a warning section in the properties page that you can drill into, as shown in Figure 13-36.

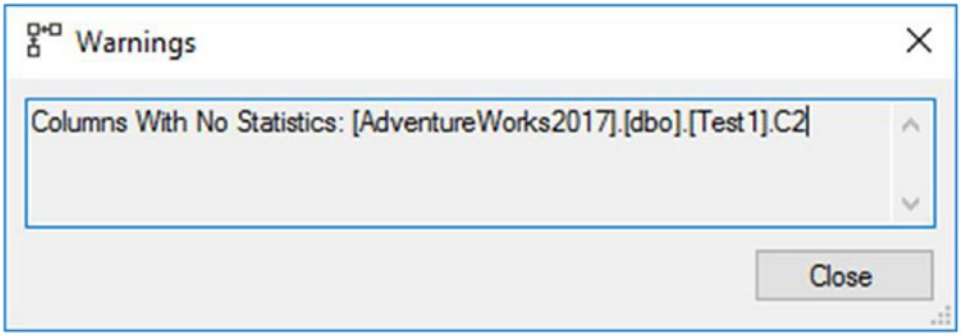


Figure 13-36. *Property values from the warning in the Index Scan operator*

Figure 13-36 shows that the statistics for the column are missing. This may prevent the optimizer from selecting the best processing strategy. The current cost of this query, as recorded by Extended Events is 100 reads and 850mc on average.

To resolve this missing statistics issue, you can create the statistics on column `Test1.C2` by using the `CREATE STATISTICS` statement.

```
CREATE STATISTICS Stats1 ON Test1(C2);
```

Before rerunning the query, be sure to clean out the procedure cache because this query will benefit from simple parameterization.

```
DECLARE @Planhandle VARBINARY(64);

SELECT @Planhandle = deqs.plan_handle
FROM sys.dm_exec_query_stats AS deqs
    CROSS APPLY sys.dm_exec_sql_text(deqs.sql_handle) AS dest
WHERE dest.text LIKE '%SELECT  *
FROM    dbo.Test1
WHERE   C2 = 1;%'

IF @Planhandle IS NOT NULL
BEGIN
    DBCC FREEPROCCACHE(@Planhandle);
END
GO
```

Caution When running the previous query on a production system, using the `LIKE '%...%'` wildcards can be inefficient. Looking for a specific string can be a more accurate way to remove a single query from the plan cache.

Figure 13-37 shows the resultant execution plan with statistics created on column `C2`.

```
Reads: 34
Duration: 4.3 ms.
```

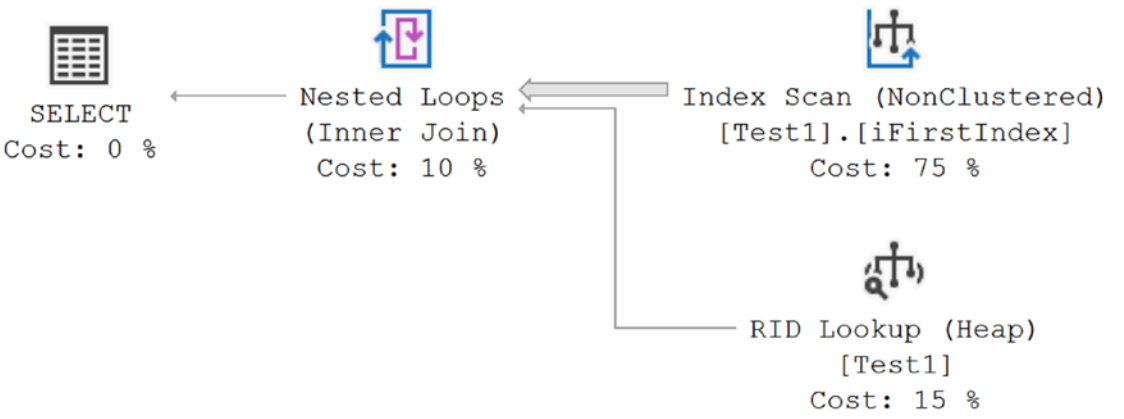


Figure 13-37. *Execution plan with statistics in place*

The query optimizer uses statistics on a noninitial column in a composite index to determine whether scanning the leaf level of the composite index to obtain the RID lookup information will be a more efficient processing strategy than scanning the whole table. In this case, creating statistics on column `C2` allows the optimizer to determine that instead of scanning the base table, it will be less costly to scan the composite index on (`C1`, `C2`) and bookmark lookup to the base table for the few matching rows. Consequently, the number of logical reads has decreased from 100 to 34, but the elapsed time has increased significantly because of the extra processing needed to join the data from two different operators.

Resolving an Outdated Statistics Issue

Sometimes outdated or incorrect statistics can be more damaging than missing statistics. Based on old statistics or a partial scan of changed data, the optimizer may decide upon a particular indexing strategy, which may be highly inappropriate for the current data distribution. Unfortunately, the execution plans don't show the same glaring warnings for outdated or incorrect statistics as they do for missing statistics. However, there is an extended event called `inaccurate_cardinality_estimate`. This is a debug event, which means its use could be somewhat problematic on a production system. I strongly caution you in its use, only when properly filtered and only for short periods of time, but I want to point it out. Instead, take advantage of Showplan Analysis detailed in Chapter 7.