第一章： An Introduction to Query Tuning and Optimization

## Architecture

SQL Server database engine are two major components:

1. The storage engine is responsible for reading data between the disk and memory in a manner that optimizes

2. The query processor accepts all queries submitted to SQL Server, devises a plan for their optimal execution, and then executes the plan and delivers the required results

The query-processing process

1. Parsing and binding

The query is parsed and bound. Assuming the query is valid the output of this phase is a logical tree, such as reading a particular table or performing an inner join

1. Query optimization

The logical tree is then used to run the query optimization process, which roughly consists of the following

two steps

1) Generation of possible execution plans

2) Cost assessment of each plan

Although the query optimizer does not generate every possible execution plan, it assesses the resource and time cost of each plan it does generate; the plan that query optimizer deems to have the lowest cost of those it has assessed is selected and then passed along to the execution engine

1. Query execution and plan caching

The query is executed by the execution engine according to the selected plan;

Parsing and Binding:

1. Parsing makes sure that T-SQL query has a valid syntax, and it translates the SQL query into an initial tree representation

2. Binding , during the binding operation, SQL Server makes sure that all the object names do exist, and it associates every table and column name on the parse tree with corresponding object in the system catalog. The output of this second process is called an algebrizer tree, which is then sent to the query optimizer.

Query Optimization:

The next step is the optimization process, which is basically the generation of candidate execution plans and the selection of the best of these plans according to their cost.AS has already been mentioned, the SQL Server query optimizer uses a cost-estimation model to

estimate the cost of each of the candidate plans

Query optimization could be also seen as the process of mapping the logical query operations expressed in the original tree representation

to physical operations, which can be carried out by the execution engine. So it's actually the functionality of the execution engine that is being implemented in the execution plans being created by the query optimizer;

Thus, the end product of the query optimization process is an execution plan: a tree consisting of number of physical operators, which contain the algorithms to be performed by the execution engine in order to obtain the desired results from the database

Generating Candidate Execution Plans:

AS stated, the basic purpose of the query optimizer is to find an efficient execution plan for your query.

The job of the query optimizer is to create and assess as many candidate execution plans as possible, with in certain criteria, in order to find a good enough plan, which may be, but it is not necessarily, the optimal plan. Theoretically, in order to find the optimum execution plan for a query. However, some complex queries may have thousands, or even millions, of possible execution plans, and although the SQL

Server query optimizer can typically consider a large number of candidate execution plans, it cannot perform an exhaustive search of all the possible plans for every query.

The query optimizer must strike a balance between optimization time and plan quality.

In order to explore the search space, the query optimizer uses transformation rules and heuristics. The generation of candidate execution plans is performed inside the query optimizer using transformation rules, and the use of heuristics limits the number of choices considered by the query optimizer is referred to as the plan space, and these plans are stored in memory during the optimization process in a component called the Memo.

Assessing the Cost of Each Plan:

Searching or enumerating candidate plans is just one part of optimization process. The query optimizer still needs to estimate the cost of these plans and select the least expensive one. To estimate the cost of a plan, it estimates the cost of each physical operator in that plan, using costing formulas that consider the use of resources such as I/O, CPU, and memory. This cost estimated depends mostly on both the algorithm used by the physical operator and the estimated number of records is known as the cardinality estimation. (基数估计)

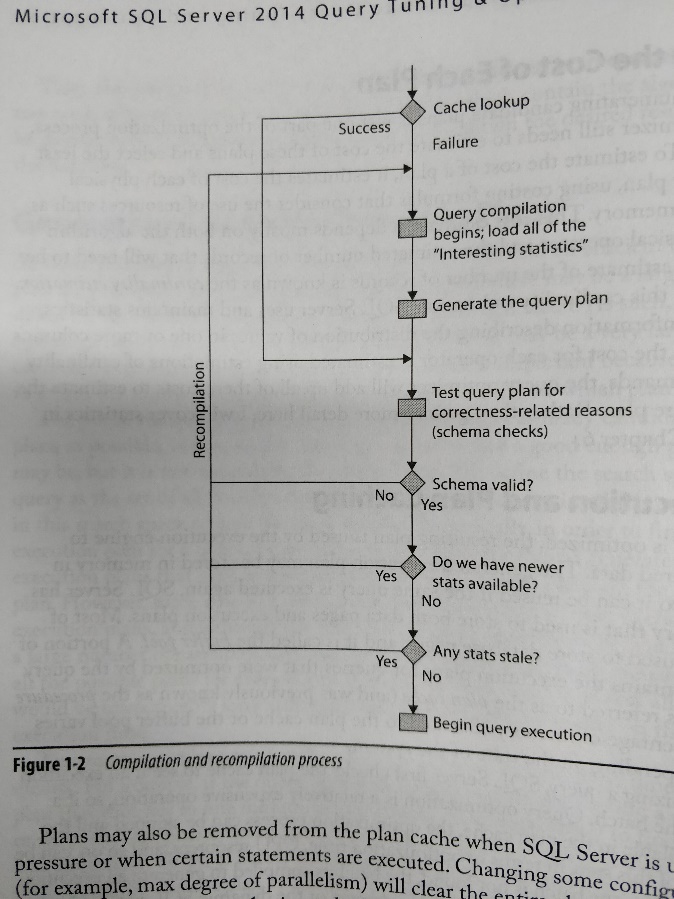
To help with this cardinality estimation, SQL Server uses and maintains statistics, which contain information describing the distribution of values in one or more columns of table. Once the cost for each operator is estimated using estimations of cardinality and resource demands, the query optimizer will add up of these costs to estimate the cost for the entire plan.

Query Execution and Plan Caching:

Once the query is optimized, the resulting plan is used by the execution engine to retrieve the desired data. The generated execution plan may be stored in memory in the plan cache so it can be reused if the same query is executed again. SQL Server has a pool of memory is used to store both data pages and execution plans. Most of this memory is used to store database pages, and it is called the buffer pool. A portion of this memory contains the execution plans for queries that were optimized by the query optimizer and is referred to as the plan cache. The percentage of memory allocated to the plan cache or the buffer pool varies dynamically, depending on the state of the system.

Before optimizing a query, SQL Server first checks the plan cache to see if an execution plan exists for the batch. Query optimization is a relatively expensive operation, so if a valid plan is available in the plan cache, the optimization process can be skipped and the associated cost of this step, in terms of optimization time, CPU resources, and so on, can be avoided. If a plan for the batch is not found, the batch is compiled to generate an execution plan for all queries in the stored procedure, the trigger, or the dynamic SQL batch. Query optimization begins by loading all the interesting statistics, when using the statistics default options, it will update the statics and will proceed with the optimization.

After a plan is found in the plan cache or a new one is created, the plan is validated for schema and data statistics changes. Schema changes are verified for plan correctness.



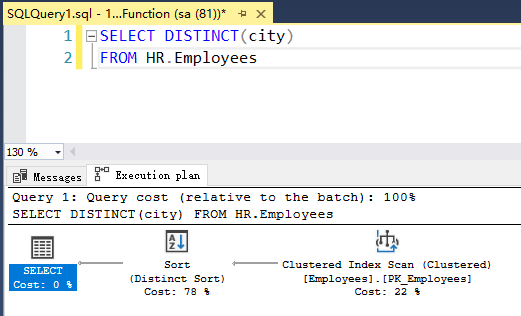
Plans may also be removed from the plan cache when SQL Server is under memory pressure or when certain statements are executed. Changing some configuration options (for example, max degree of parallelism) will clear the entire plan cache. Alternatively, some statements, such as altering a database with certain ALTER DATABASE options, will clear all the plans associated with that particular database.

However, it is also worth noting that reusing an existing plan may not always be the best solution for a given query, and some problems may appear. For Example, depending on the data distribution within a table, the optimal execution plan for a query may differ greatly depending on the parameters being used.

## Execution Plans

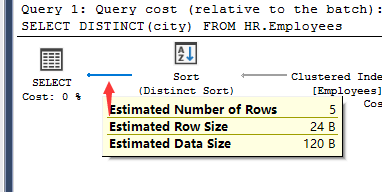
Now that we have a foundation in the query processor and how it works its magic, it’s time to consider how we can interact with it. Primarily, we’ll interact with the query processor through execution plans, which as I mention

Graphical Plans



Each node in the tree structure is represented as an icon that specifies a logical and physical operator.

Each node is related to a parent node, connected with arrowheads, where data flows from a child operator to a parent operator and the arrow width is proportional to the number of rows. After the operator performs some function on the records it has read, the results are output to its parent.



To perform their job, physical operators implement at least the following three methods:

\* Open() Causes an operator to be initialized, and may include setting up any required data structures

\* GetRow() Requests a row from the operator

\* close() Performs some cleanup operations and shuts down the operator once it has performed

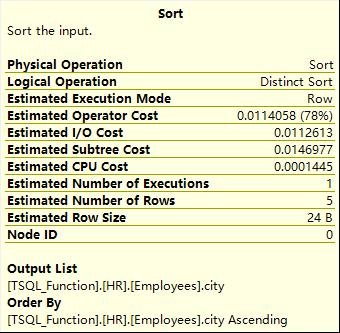
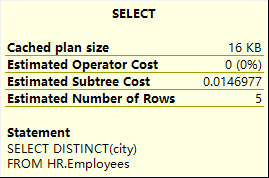
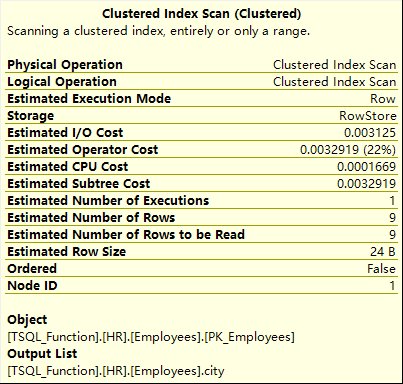
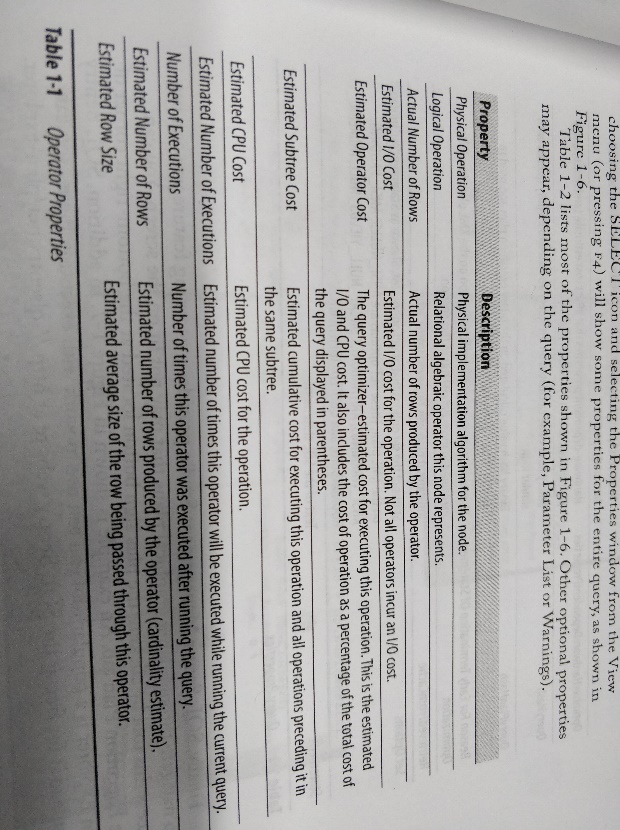
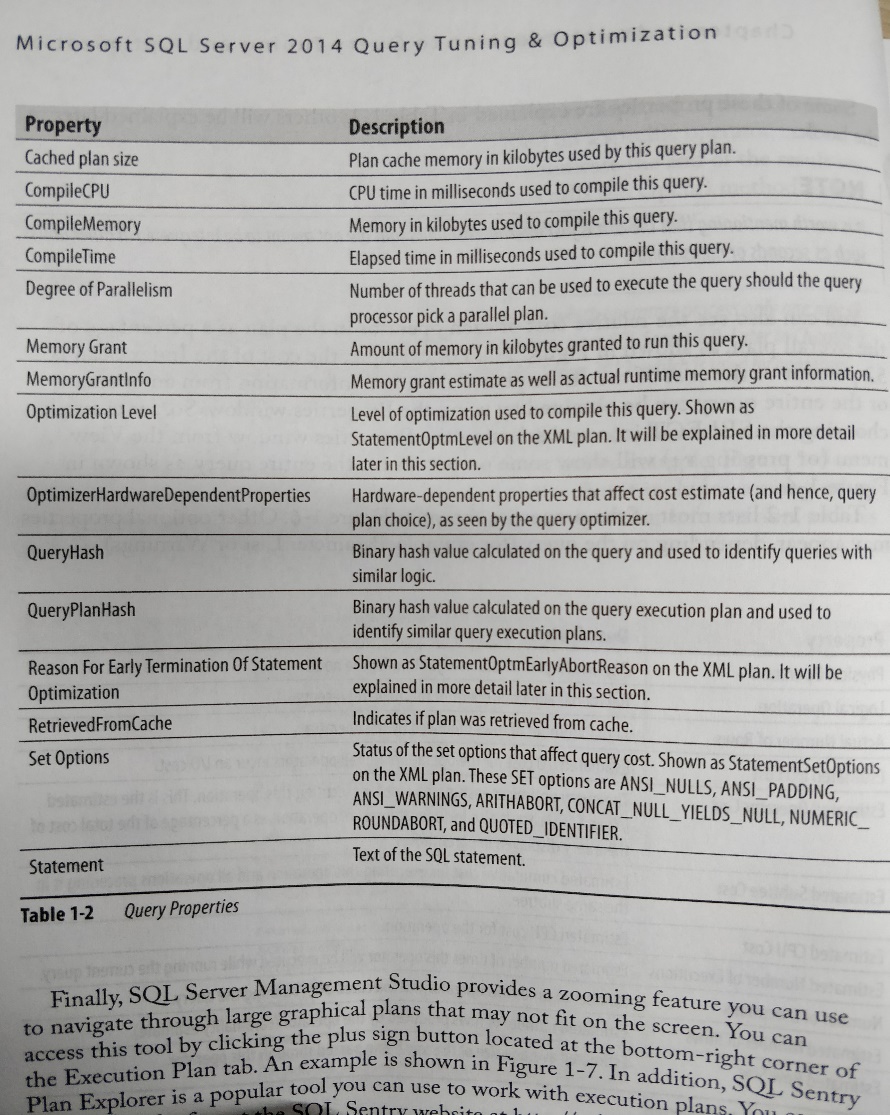
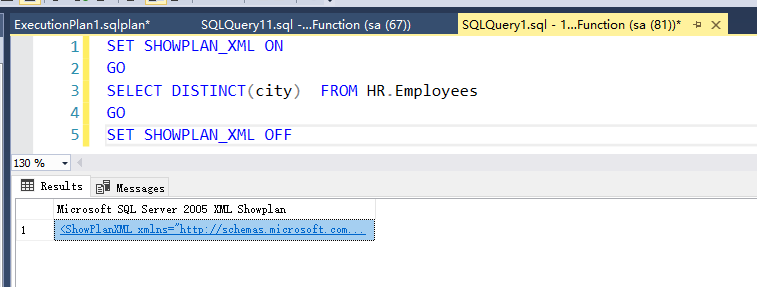
 

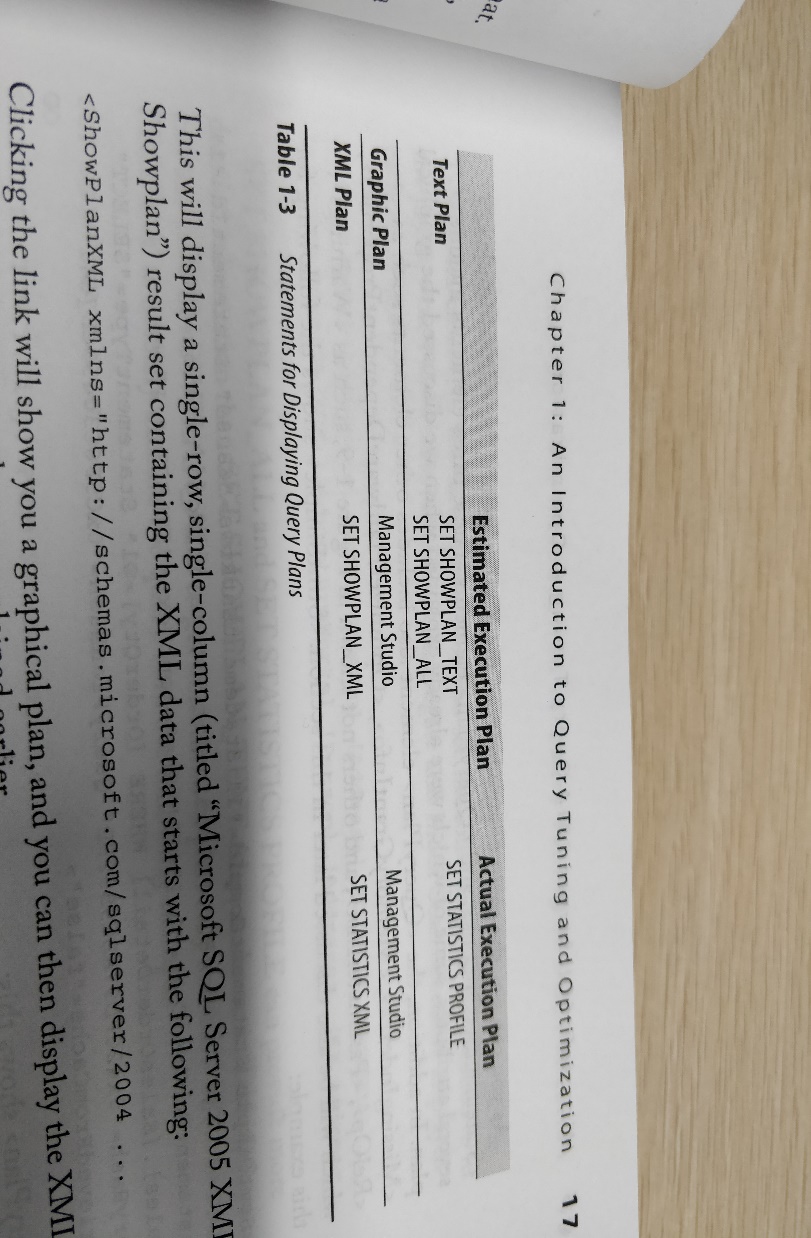
Table1-1 Operator Properties





XML





Text Plans

SET SHOWPLAN\_TEXT ON

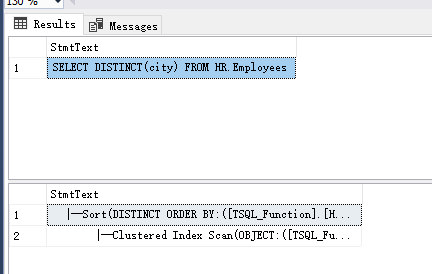
GO

SELECT DISTINCT(city) FROM HR.Employees

GO

SET SHOWPLAN\_TEXT OFF

GO



SET SHOWPLAN\_ALL ON

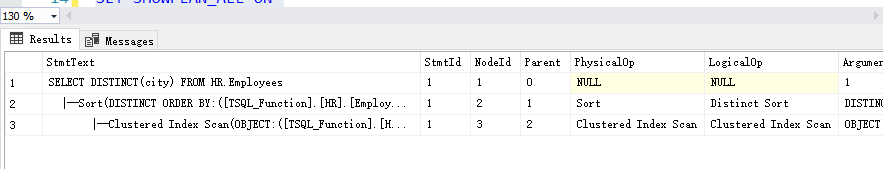
GO

SELECT DISTINCT(city) FROM HR.Employees

GO

SET SHOWPLAN\_ALL OFF

GO



SET STATISTICS PROFILE ON

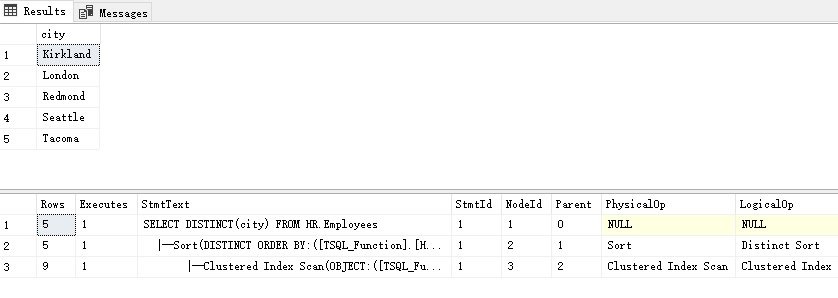
GO

SELECT DISTINCT(city) FROM HR.Employees

GO

SET STATISTICS PROFILE OFF

GO



Additional Plan Properties

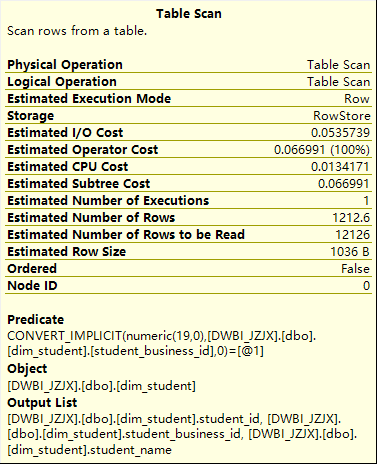
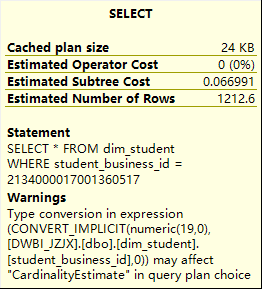
More detail in Chapter 4. Let’s start with the StatementOptmLevel, StatementOptmEarlyAbortReason, and CardinalityEstimationModelVersion attributes of the <StmtSimple> element.

StatementOptmLevel is the query optimization level, which can be either TRIVIAL or FULL. The optimization process may be expensive to initialize and run for very simple queries that don’t require any cost estimation, so to avoid this expensive operation for these simple queries, SQL Server uses the trivial plan optimization. If a query does not apply for a trivial optimization, a full optimization will have to be performed.

The following query will produce a trivial plan:

SELECT \* FROM dim\_student

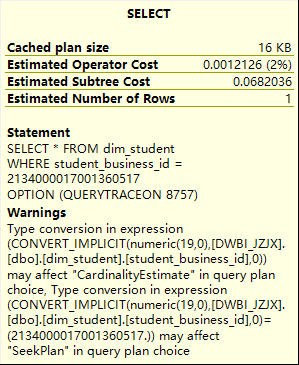
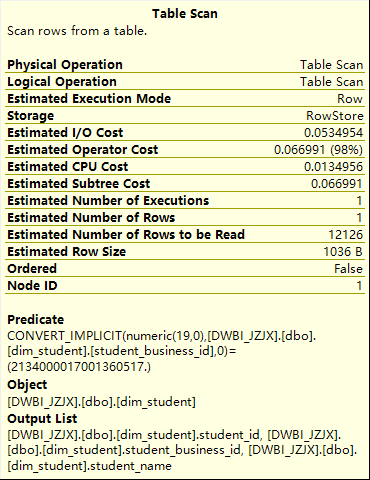
WHERE student\_business\_id = 2134000017001360517



SELECT \* FROM dim\_student

WHERE student\_business\_id = 2134000017001360517

OPTION (QUERYTRACEON 8757)

The QUERYTRACEON query hint is used to apply a trace flag at the query level. After running the previous query, SQL Server will run a full optimization, which you can verify with StatementOptmLevel as FULL in the result plan.

(参考：https://dotblogs.com.tw/ricochen/2016/09/28/205550)

备注：

Trace Flag是做什么的？

Trace Flage 是暂时关闭或设定SQL Server 特定行为。例如，启动SQL　Server 并设定3226,就会关闭将每项成功的备份作业写入SQL Server错误记录档和系统事件记录。如果备份频率高（交易记录），那么这些成功讯息可能快速累积。

进而产生大量的错误记录，一来难以搜索，二来也多少影响SQL Server 读取该档案效能。

Warnings on Execution Plans

The SQL Server 2012 show plan schema added six more iterator – or query-specific warnings:

* SpillToTempDb
* Wait
* PlanAffectingConvert
* SpatialGuess
* UnmatchedIndexes
* FullUpdateForOnlineIndexBuild

ColumnsWithNoStatistics

This warning means that the query optimizer tried to use statistics but none were available.

DROP STATISTICS Sales.OrderDetails.\_WA\_Sys\_0000000C\_49C3F6B7

Next, temporarily disable automatic creation of statistics at the database level:

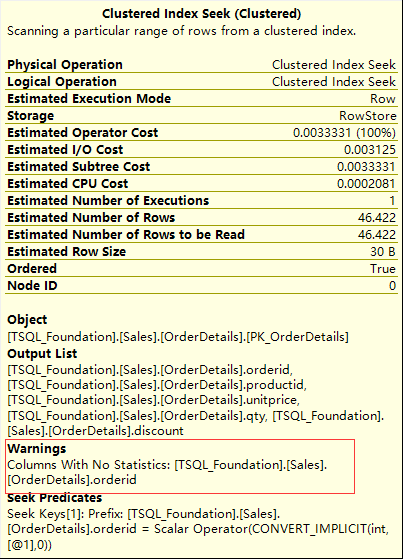
ALTER DATABASE AdventureWorks2012 SET AUTO\_CREATE\_STATISICS OFF

SELECT \* FROM Sales.OrderDetails

WHERE orderid = 10623

最后不要忘记

ALTER DATABASE AdventureWorks2012 SET AUTO\_CREATE\_STATISICS ON



NojoinPredicate

A possible problem while using the old-style ANSI SQL-89 join

SELECT \* FROM Sales.SalesOrderHeader soh, Sales.SalesOrderDetail sod

WHERE soh.SalesOrderID = sod.SalesOrderID

PlanAffectingConvert

This warning shows that type conversions were performed that may impact the performance of the resulting execution plan.