[1. 自定义函数和存储过程的区别是什么，什么情况下只能用自定义函数，什么情况下只能用存储过程？ 4](#_Toc467248446)

[1-1. 调用函数时，默认参数不能省 4](#_Toc467248447)

[2. Group by 和Case 语句 5](#_Toc467248448)

[3. Row\_Number（）的使用 5](#_Toc467248449)

[3-1. 返回row number 6](#_Toc467248450)

[3-2. 使用 row number 来返回分页数据 6](#_Toc467248451)

[3-3. 在row number 中使用分组 7](#_Toc467248452)

[4. 使用sql 找出当月的天数 7](#_Toc467248453)

[5. Cast vs Convert 7](#_Toc467248454)

[5-1. Cast 在sql server 内部其实还是通过convert 实现的 8](#_Toc467248455)

[5-2. 转换为char，varchar 时，不指定长度的话，默认是30 9](#_Toc467248456)

[6. SQL 优化 9](#_Toc467248457)

[7. 插入10万条数据大概需要多少时间 9](#_Toc467248458)

[8. 查询sql server 的版本 10](#_Toc467248459)

[8-1. 14的版本 10](#_Toc467248460)

[8-2. 12的版本 10](#_Toc467248461)

[8-3. 2008的版本 10](#_Toc467248462)

[8-4. 2005的版本 11](#_Toc467248463)

[8-5. 2000的版本 11](#_Toc467248464)

[9. 索引和数据的存储 11](#_Toc467248465)

[10. 查询计划 13](#_Toc467248466)

[10-1. 基本概念 13](#_Toc467248467)

[10-1-1. 阻断运算和非阻断运算 13](#_Toc467248468)

[10-2. 查看执行计划 14](#_Toc467248469)

[10-2-1. 有了图形化的形式，为什么还要文本和XML 15](#_Toc467248470)

[10-2-2. XML格式的查询计划是SQL Server 存储时用的格式 15](#_Toc467248471)

[10-2-3. Set Statistics profile on ：以文字格式显示执行计划 15](#_Toc467248472)

[10-3. 常见查询计划里的操作 15](#_Toc467248473)

[10-3-1. 连接(Join)和联合(Union) 15](#_Toc467248474)

[10-3-2. Index Seek 16](#_Toc467248475)

[10-3-3. Bookmark Lookup （书签查找）/Key Lookup 16](#_Toc467248476)

[10-3-4. Index Seek 和 Bookmark lookup 16](#_Toc467248477)

[10-3-5. Table Scan 有 执行次数大于1的么？ 16](#_Toc467248478)

[10-3-6. 聚合操作并不一定非要先sort 16](#_Toc467248479)

[10-3-7. Spool 17](#_Toc467248480)

[10-4. 仅仅是参数不同就有可能产生不同的执行计划 18](#_Toc467248481)

[10-4-1. 同样的query， 数据不同时，执行计划也可能不同 18](#_Toc467248482)

[10-4-2. 硬件资源不同时，也可能会生成不同的执行计划 19](#_Toc467248483)

[10-5. 看懂SqlServer查询计划 19](#_Toc467248484)

[10-6. 逻辑运算符和物理运算符参考 19](#_Toc467248485)

[10-7. 用于对运行慢的查询进行分析的清单 19](#_Toc467248486)

[10-8. The Myth About Estimated Execution Plans 20](#_Toc467248487)

[10-9. 子查询 20](#_Toc467248488)

[10-9-1. 独立的子查询 20](#_Toc467248489)

[10-9-2. 相关的子查询 20](#_Toc467248490)

[11. 统计信息 20](#_Toc467248491)

[11-1. 更新统计信息 20](#_Toc467248492)

[11-1-1. 使用Visual Studio 20](#_Toc467248493)

[11-1-2. 使用SQL 20](#_Toc467248494)

[11-1-3. 慎用sp\_updatestats 21](#_Toc467248495)

[11-2. 统计信息的失效和更新 21](#_Toc467248496)

[11-2-1. 统计信息会失效 21](#_Toc467248497)

[11-2-2. 统计信息会不会自动更新取决于一些配置 22](#_Toc467248498)

[11-3. 查询优化器在执行查询之前会先check 统计信息是否已经失效 22](#_Toc467248499)

[11-4. 有了自动更新，为什么查询优化时还会建议手动更新统计信息 22](#_Toc467248500)

[12. 性能调优和查询优化 23](#_Toc467248501)

[12-1. Performance Tuning VS Query Optimization 23](#_Toc467248502)

[12-2. SQL Server调优系列进阶篇（深入剖析统计信息） 23](#_Toc467248503)

[12-3. 注意操作执行的次数 23](#_Toc467248504)

[12-4. 注意返回的记录条数 23](#_Toc467248505)

# 自定义函数和存储过程的区别是什么，什么情况下只能用自定义函数，什么情况下只能用存储过程？

存储过程：功能远强于函数，可以执行包括修改表等一系列数据库操作，可以由外部应用程序进行调用或在SQL中调用。

以下两种情况下只能用存储过程，而不能用自定义函数 ：1.要执行DML,修改数据。2.要执行动态SQL语句,执行任何存储过程。

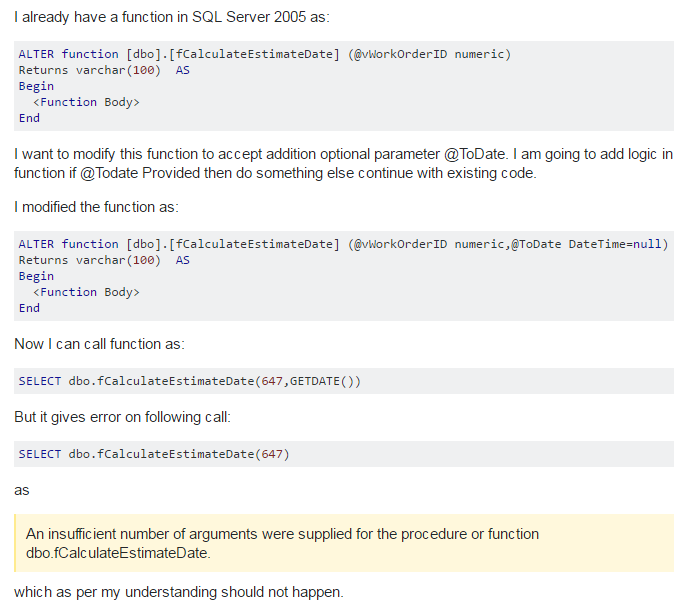
存储过程，其返回值不能被直接引用。 自定义函数，其返回值可以被直接引用。

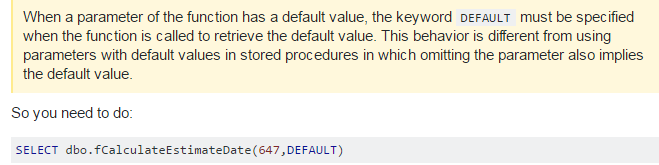
存储过程，可返回记录集。 自定义函数，可以返回表变量。

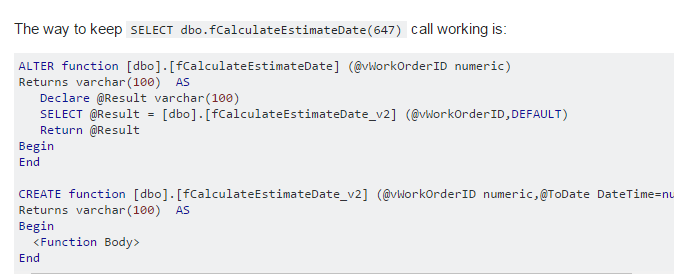
## 调用函数时，默认参数不能省

原先有一个参数，想的是给加一个默认参数，这样旧有逻辑不需要动，新逻辑传入这个参数就行。存储过程已经这么做过多次了

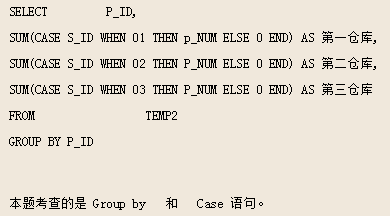
但是在执行的时候，旧有逻辑报错，说函数的参数数目不对，一查原来是调用的时候不能省啊，必须传入default才行。 要不就加一个新函数，在旧函数里默认调用这个新函数







# Group by 和Case 语句

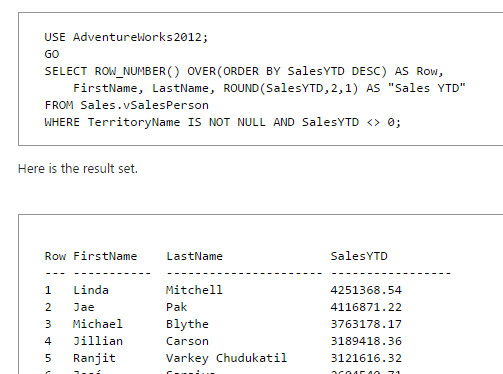


# Row\_Number（）的使用

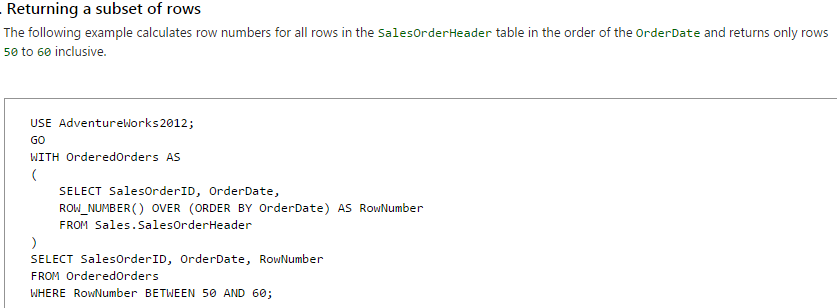
Row\_number() 必须和over 一起使用， over 里面必须至少有一个order by 语句，可以可选一个分组语句



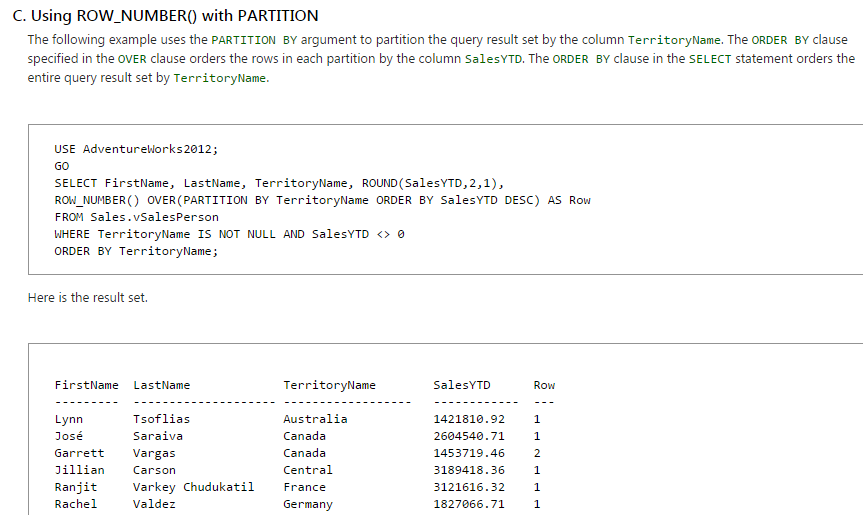
## 返回row number



## 使用 row number 来返回分页数据



## 在row number 中使用分组



# 使用sql 找出当月的天数

Can you use a batch SQL or store procedure to calculating the Number of Days in a Month

select datepart(dd,dateadd(dd,-1,dateadd(mm,1,cast(cast(year(getdate()) as varchar)+'-'+cast(month(getdate()) as varchar)+'-01' as datetime))))

用getdate() 取得当天日期

用year() 取得当天年， month()取得当天月， 加上-01 构成当月的第一天。

用cast 转换为datetime

用dateadd(mm,1, datetime) 转换为下个月的第一天

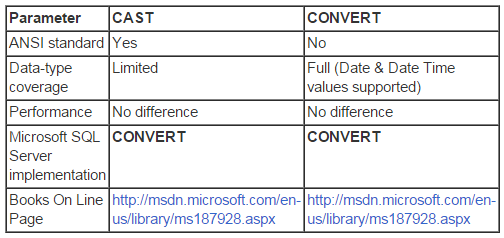
用dateadd(dd,-1) 转换为前月最后一天

用datepart(dd, datetime)来取得里面的天数

# Cast vs Convert

两个函数不存在性能上的差异。只有语法和是否ANSI的差异。然后Convert在处理datetime，money等的时候功能更强大一些。

prefer to go with CAST unless I am converting dates or date-time values



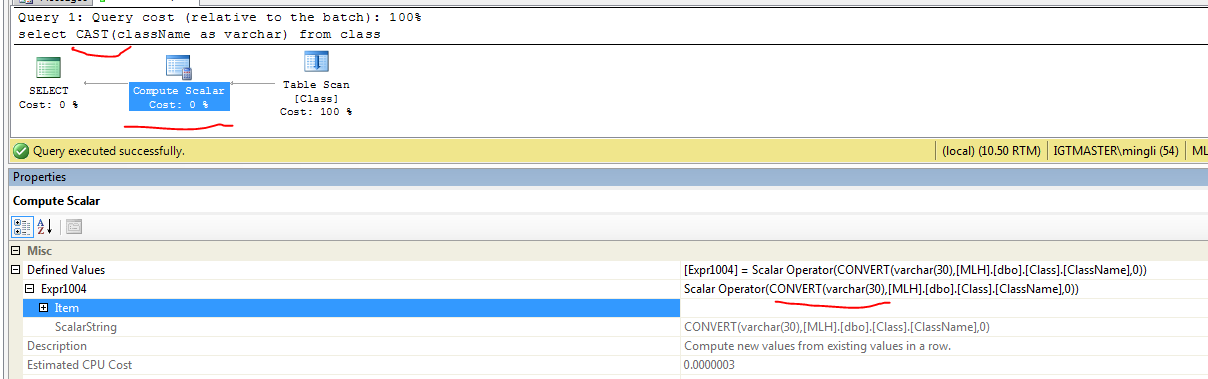
CONVERT is specific to SQL Server, and allows for a greater breadth of flexibility when converting between date and time values, fractional numbers, and monetary signifiers.

CAST is the more ANSI-standard of the two functions, meaning that while it's more portable, it's also less powerful. CAST is also required when converting between decimal and numeric values to preserve the number of decimal places in the original expression. For those reasons, it's best to use CAST first, unless there is some specific thing that only CONVERT can provide in the work you're doing.

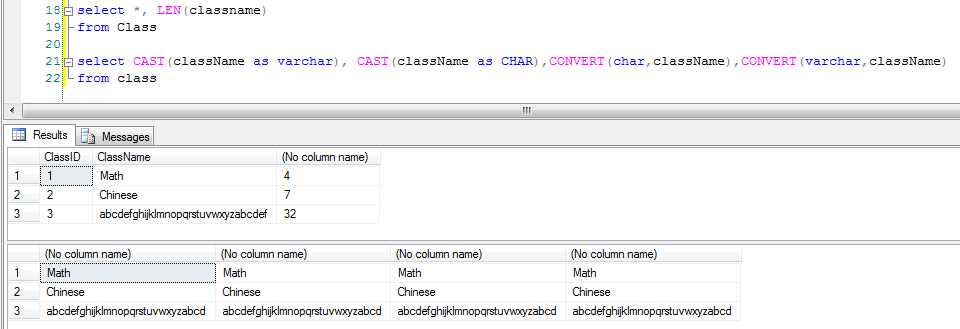
有时候结合这两个一起使用会更灵活。

SELECT CAST(CONVERT(CHAR(10),CURRENT\_TIMESTAMP,102) AS DATETIME

## Cast 在sql server 内部其实还是通过convert 实现的



## 转换为char，varchar 时，不指定长度的话，默认是30



多出来的话会被截掉

# SQL 优化

<http://www.blogjava.net/allen-zhe/archive/2010/07/23/326927.html>

<http://www.jb51.net/article/16026.htm>

<http://blog.csdn.net/dinglang_2009/article/details/8682412>

<http://www.cnblogs.com/lzrabbit/archive/2012/05/06/2485572.html>

<http://www.cnblogs.com/gaizai/archive/2012/01/20/2327814.html>

# 插入10万条数据大概需要多少时间

CREATE TABLE Employee (

EmployeeId NVARCHAR(32) NOT NULL PRIMARY KEY,

EmployeeName NVARCHAR(40) NOT NULL,

);

SET NOCOUNT ON

declare @i int

set @i=1

while @i<=100000

begin

INSERT INTO Employee VALUES(replace(newid(), '-', ''), 'Employee\_' + CONVERT(varchar, @i) );

set @i = @i+1

end

在本机测试一共用了49分21秒，即2961秒。每条记录的插入时间为0.02961秒，合29.61毫秒，约30毫秒

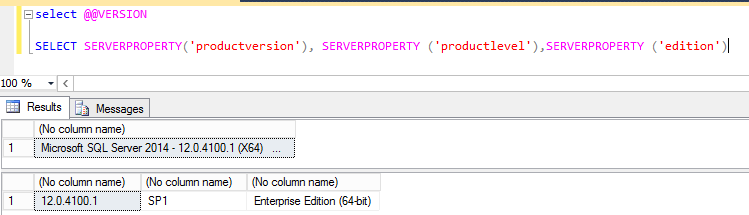
# 查询sql server 的版本

<https://support.microsoft.com/zh-cn/kb/321185>

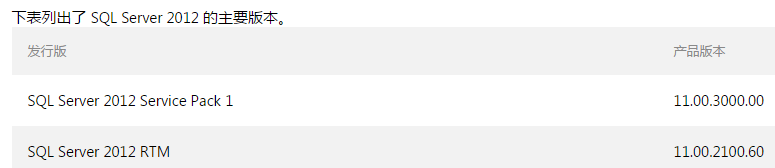




## 14的版本



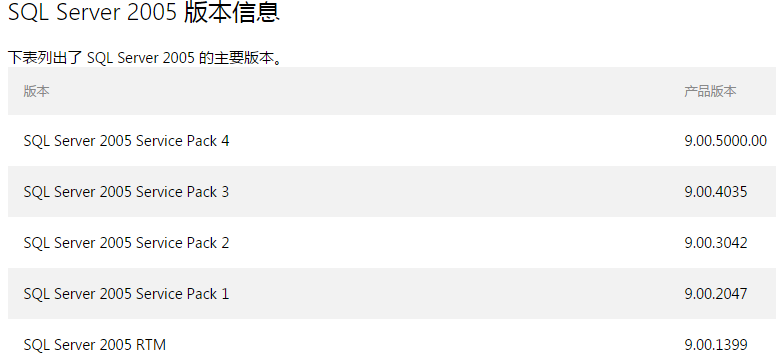
## 12的版本



## 2008的版本



## 2005的版本



## 2000的版本



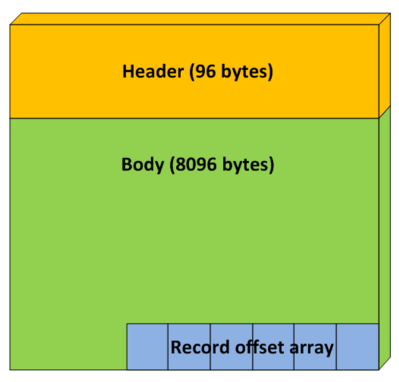
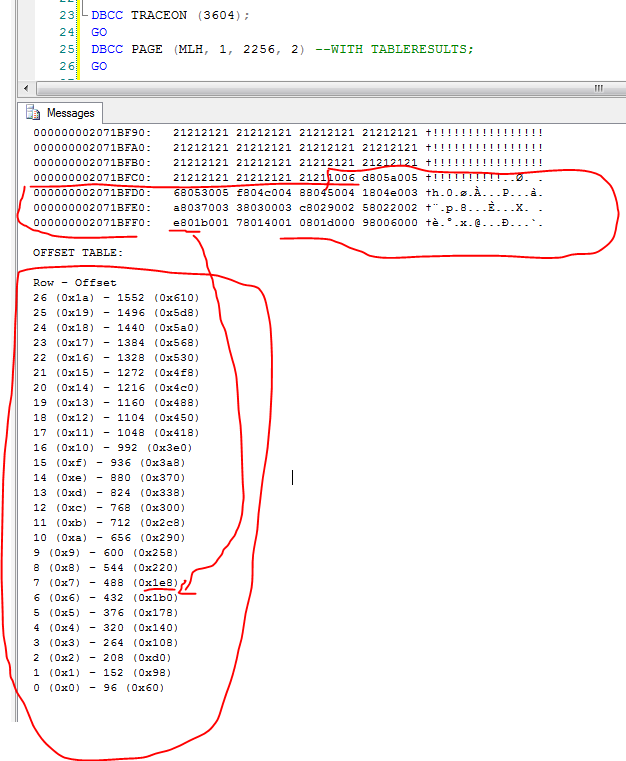
# 索引和数据的存储

[数据结构和存储结构](索引数据结构/数据结构和存储结构.pdf)

[SQL Server Storage Internals 101](索引数据结构/SQL%20Server%20Storage%20Internals%20101.pdf)

1. Sql server用页（Page）来存储数据，每一页的大小都是8K, 不管是索引页还是数据页，都是8K
2. Sql server的索引用B+ 树来存储： 这句话的意思是， 存储索引数据的那些页以B+树的结构来组织。每一页都是B+树上的一个节点（在这个上下文里，节点和页是一个意思）。 即节点的数据结构就是页，**而不是像以前理解的那样，节点有自己的数据结构，然后节点再存储在页里，这样就可以一个页存储着好几个节点的数据，然后节点链成B+数。**实际是一个节点就是一个页，一个页同子页之间的链接是通过页里面的索引行中的页指针来实现的。一个索引页有几个子页，就有几个索引行。
3. 每个页的数据结构都是一样的，前面是96 bytes的Header， 剩下的是body。 Body的最后面是行的偏移数组。记录的是各行相对于页开始位置0 的偏移量。比如第一行的偏移量是0x60 ，即偏移96。 其他的取决于每一行的长度。 第一行的偏移量在最后位置，然后从右往左分别是第二行，第三行

如下图就是看到的26个行的偏离量和它在数据页的位置（偏移数组在页的最末尾，第一行的偏移在偏移数组的最末尾）每个偏移量为16位两个字节

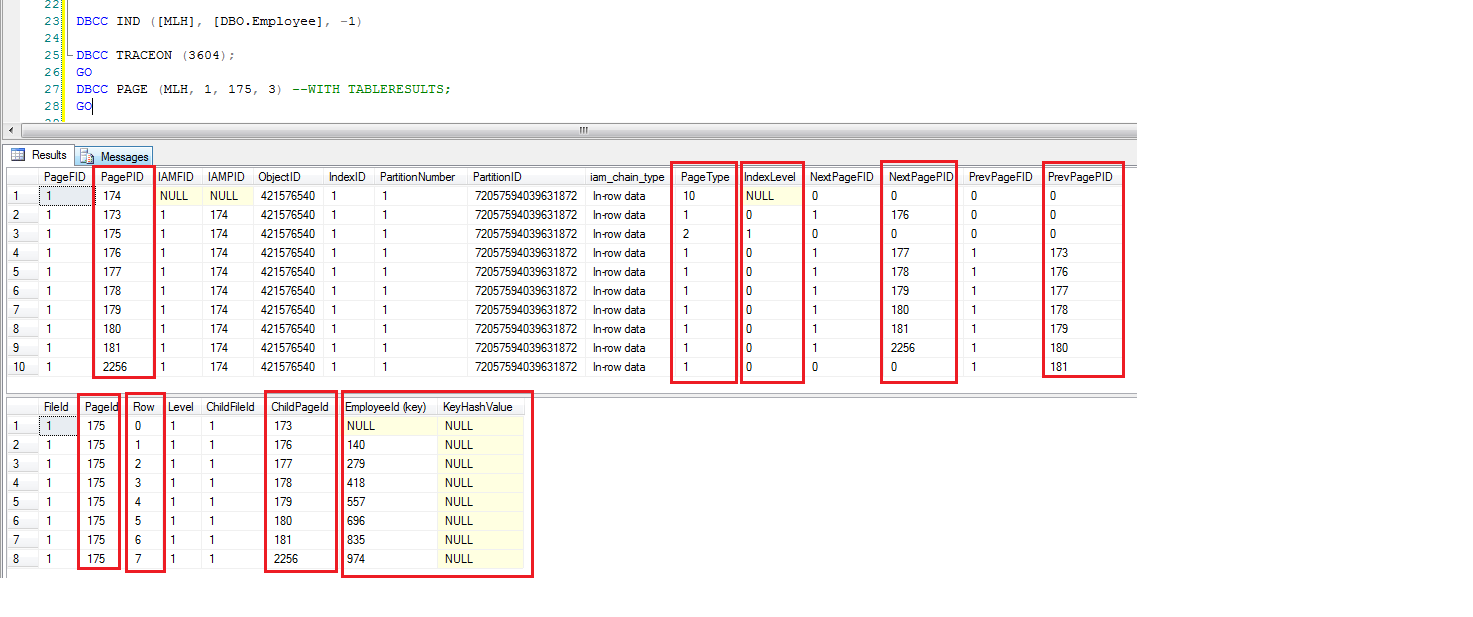
1. 不管是数据页还是索引列， 里面的数据或者索引都是以行结构来存储的。对于数据页来说，每一行就是普通的一行数据，里面包含各个列的数据。对于索引行来说，每一行的数据包括索引的字段（联合索引的话应该会存索引中的所有的字段），（主键信息字段 待定，或者是对应行的hash 信息）， 以及指向下一级页节点的指针。
2. 利用 DBCC IND 和 DBCC PAGE 可以构建出 整个B+数，上面是索引节点，下面是叶节点。对于聚集索引来说，叶节点就是物理的数据。对于非聚集索引来说，页节点是指向为例数据的行标签。

[聚集索引结构](索引数据结构/聚集索引结构.pdf)

叶子节点的IndexLevel 是0， 然后从叶子到中间节点到根节点依次递增。

NextPagePID和 PrePagePID 只的是同层次之间各个页面之间的链接关系（兄弟节点）。可以看到是双链表。而页面同子节点之间的链接是在数据或者索引行中出现的。

PageType : 1 是数据页 2 是索引页 10 是IAM页



[B+树](索引数据结构/B+树.xlsx)

1. 所以，如果拿字典举例子的话，他前面的按拼音的目录页加上后面真正的字典页一共构成了聚集索引。 即数据也是聚集索引的一部分。而那些按偏旁部首的索引则为非聚集索引。
2. 由于数据是按聚集索引的顺序而物理排序的，所以一个表中只能有一个聚集索引。
3. 索引一定是排过序的
4. 通过非聚集索引间接找数据，如果数据条数比较少，性能还可以。如果是以万级别，十万级别的行，那应该还没有一行行全表扫描来得快。

# 查询计划

## 基本概念

### 阻断运算和非阻断运算

所谓阻断和非阻断的区别就是：运算符是否在输入数据的时候能够直接输出结果数据。

a、当一个运算符在消耗输入行的同时生成输出行，这种运算符就是非阻断式的。

比如我们经常使用的 Select Top …操作，此操作就是输入行的同时进行输出行操作，所以此操作就是非阻断式的。

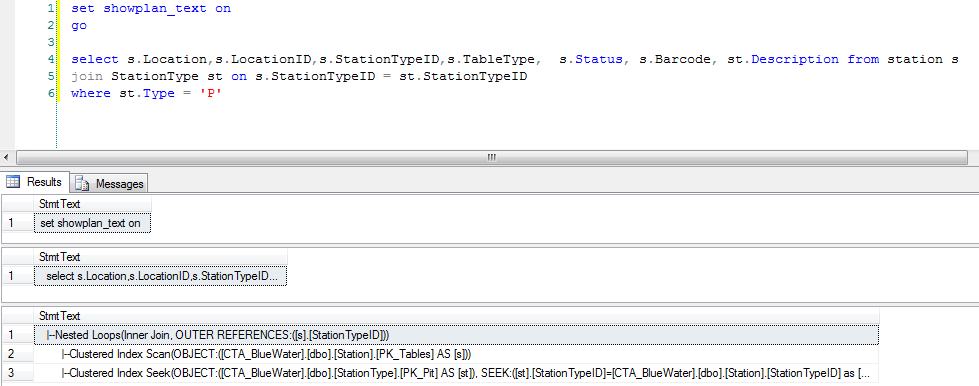
b、当一个运算符所产生的输出结果需要等待所有的数据输入的时候，这个操作运算就是阻断运算的。

比如上面我们举的例子Count(\*)操作，此操作就需要等待所有的数据行输入才能计算出，所以为阻断式运算，另外还有分组计算。

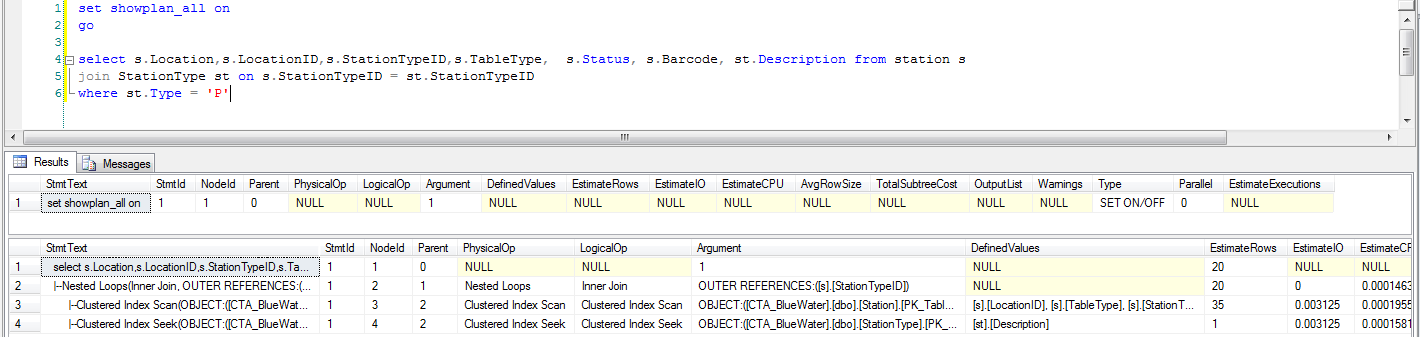
提示：并不是所有的阻断式操作就需要消耗内存，比如Count(\*)就为阻断式，但它不消耗内存，但大部分阻断式操作都会消耗内存。

在大部分的OLTP系统中，我们要尽量的使用非阻断式操作来代替阻断式操作，这样才能更好的提高相应时间，比如有时候我们用EXISTS子查询来判断，比用SELECT count(\*)>0的速度要理想的多。

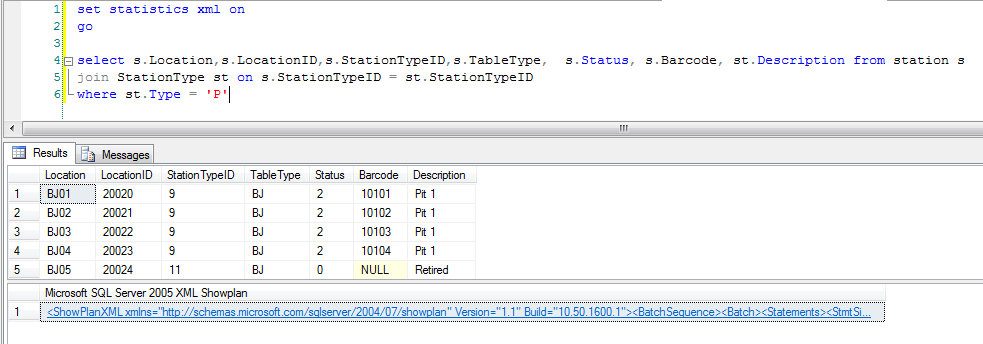
## 查看执行计划



Go 必须加上， 因为set 执行必须单独在一个代码块里



都应该是Estimated Execution Plan， 因为没有Actual 的Rows 等信息，都是EstimatedRows



### 有了图形化的形式，为什么还要文本和XML

图形化的显示依赖于SQL Server Management Studio 这个工具。

考虑不用SQL Server Management Studio， 只有一个sql server 的连接命令行的时候，如何拿到查看执行计划呢。

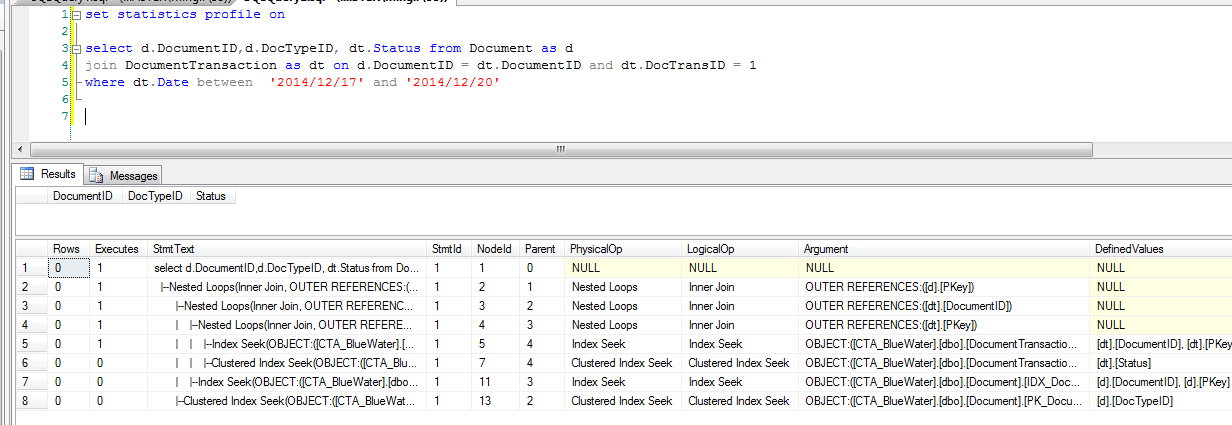
### XML格式的查询计划是SQL Server 存储时用的格式

XML方式提供的查询计划信息是最为全面的，并且在SQL Server内部存储的查询计划类型也为XML数据类型。

### Set Statistics profile on ：以文字格式显示执行计划

只要在执行之前 用set statistics profile on 语句就行。

这样就不需要勾选 Include Actual Execution Plan 来看实际的执行计划了。当然勾选了也可以，多显示一个Tab 页而已。



## 常见查询计划里的操作

### 连接(Join)和联合(Union)

注意翻译是不同的

### Index Seek

### Bookmark Lookup （书签查找）/Key Lookup

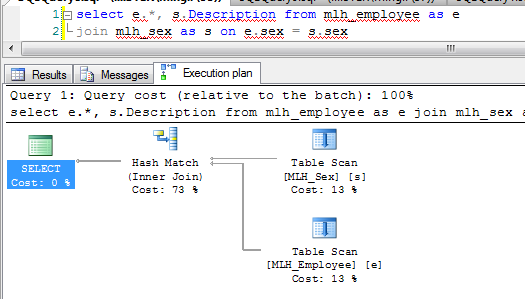
在SQL Server2005 SP2版本以上，书签查找也被称为键查找，其实是一个概念。

### Index Seek 和 Bookmark lookup

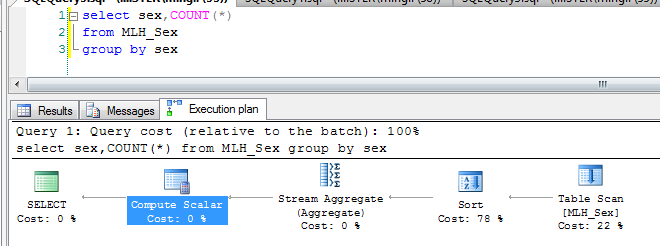
索引查找并不一定非要伴随着bookmark lookup

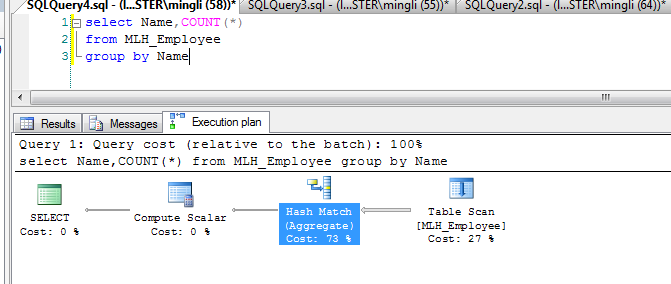
只有索引记录里找不到的列，才会再需要bookmark lookup

### Table Scan 有 执行次数大于1的么？



### 聚合操作并不一定非要先sort





### Spool

The spool operator consumes a set of data, and stores it in a temporary structure, in the tempdb database.

This structure is typically either a Table (ie, a heap (Edit: Ok, it's not actually stored as a heap, but it may as well be, because it always scans)), or an Index (ie, a b-tree). If no data is actually needed from it, then it could also be a Row Count spool, which only stores the number of rows that the spool operator consumes. A Window Spool is another option if the data being consumed is tightly linked to windows of data, such as when the ROWS/RANGE clause of the OVER clause is being used.

A Table or Index Spool is either Eager or Lazy in nature. Eager and Lazy are Logical operators, which talk more about the behaviour, rather than the physical operation.In the SQL world it describes the idea of either fetching all the rows to build up the whole spool when the operator is called (Eager), or populating the spool only as it’s needed (Lazy).

**And when is it needed?**

The way I see it, spools are needed for two reasons.

1 – When data is going to be needed AGAIN.

2 – When data needs to be kept away from the original source.

If you’re someone that writes long stored procedures, you are probably quite aware of the second scenario. I see plenty of stored procedures being written this way – where the query writer populates a temporary table, so that they can make updates to it without risking the original table. SQL does this too. Imagine I’m updating my contact list, and some of my changes move data to later in the book. If I’m not careful, I might update the same row a second time (or even enter an infinite loop, updating it over and over). A spool can make sure that I don’t, by using a copy of the data. This problem is known as the Halloween Effect (not because it’s spooky, but because it was discovered in late October one year). As I’m sure you can imagine, the kind of spool you’d need to protect against the Halloween Effect would be eager, because if you’re only handling one row at a time, then you’re not providing the protection... An eager spool will block the flow of data, waiting until it has fetched all the data before serving it up to the operator that called it.

**Table v Index**

When considering whether to use a Table Spool or an Index Spool, the question that the Query Optimizer needs to answer is whether there is sufficient benefit to storing the data in a b-tree. The idea of having data in indexes is great, but of course there is a cost to maintaining them. Here we’re creating a temporary structure for data, and there is a cost associated with populating each row into its correct position according to a b-tree, as opposed to simply adding it to the end of the list of rows in a heap. Using a b-tree could even result in page-splits as the b-tree is populated, so there had better be a reason to use that kind of structure. That all depends on how the data is going to be used in other parts of the plan. If you’ve ever thought that you could use a temporary index for a particular query, well this is it – and the Query Optimizer can do that if it thinks it’s worthwhile.

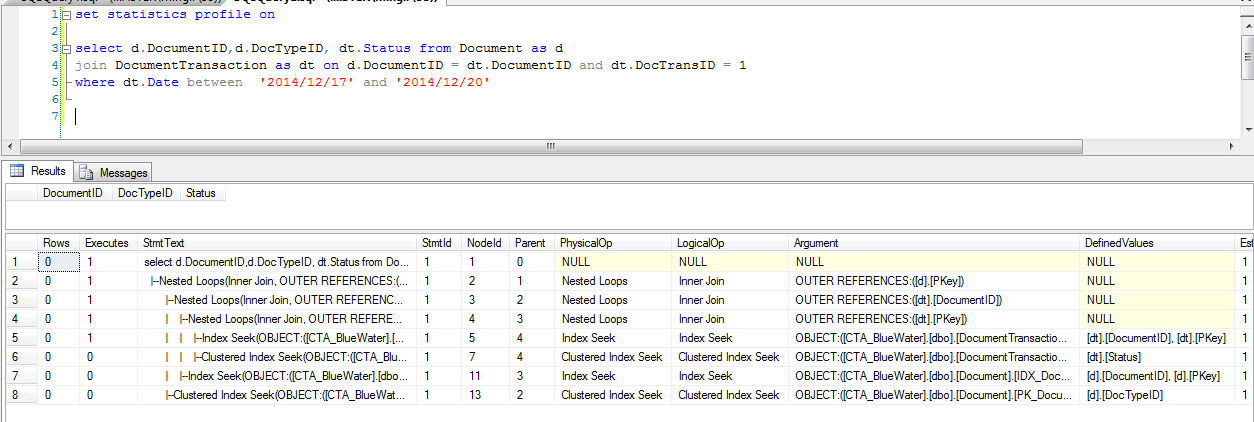
It’s worth noting that just because a Spool is populated using an Index Spool, it can still be fetched using a Table Spool. The details about whether or not a Spool used as a source shows as a Table Spool or an Index Spool is more about whether a Seek predicate is used, rather than on the underlying structure.

## 仅仅是参数不同就有可能产生不同的执行计划

后者用了一个系统预先定义好的View。

因为DocumentTransaction 表里最大的数据也就是2004年的，所以2014年的时候估计现有的索引统计信息不足以反应。

所以，查询优化时，最好的就是结合具体的数据，具体的参数来进行



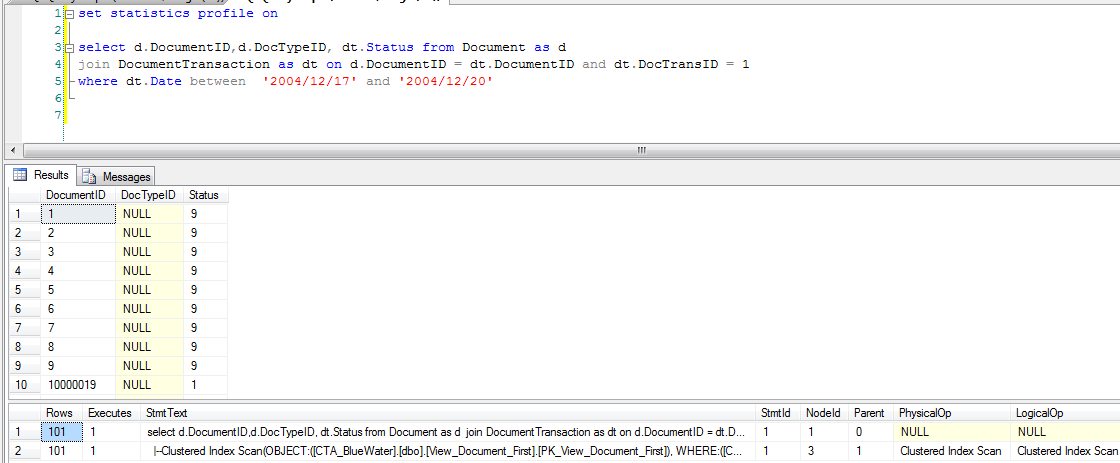
### 同样的query， 数据不同时，执行计划也可能不同

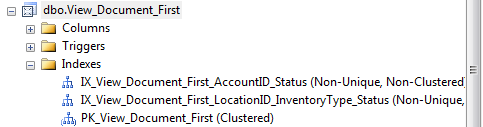
即使同样的执行语句，面对不同的数据时，也可能会有不同的执行计划。

在SQL Server中并不是固定的语句就会形成特定的计划，并且生成的特定计划也不是总是最优的，这和数据库现有数据表中的内容分布、数据量、数据类型等诸多因素有关，而记录这些详细信息的就是统计信息。

### 硬件资源不同时，也可能会生成不同的执行计划

在SQL Server根据库内容形成的统计信息进行评估的同时，还要参照当前运行的硬件资源，有时候它认为最优的方案可能当前硬件资源不支持，比如：内存限制、CPU限制、IO瓶颈等，所以执行计划的优劣还要依赖于底层硬件。





## 看懂SqlServer查询计划

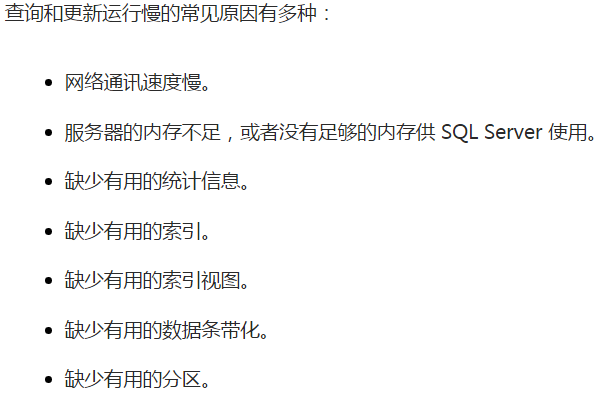
<http://www.cnblogs.com/fish-li/archive/2011/06/06/2073626.html>

## 逻辑运算符和物理运算符参考

<https://msdn.microsoft.com/zh-cn/library/ms191158(v=sql.120).aspx>

## 用于对运行慢的查询进行分析的清单

<https://msdn.microsoft.com/zh-cn/library/ms177500(v=sql.105).aspx>



## The Myth About Estimated Execution Plans

<http://sqlmag.com/sql-server/myth-about-estimated-execution-plans>

## 子查询

### 独立的子查询

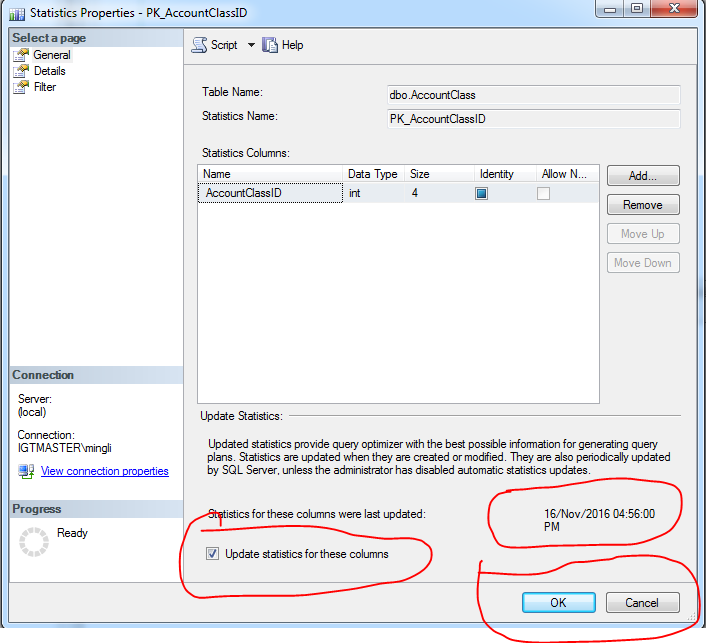
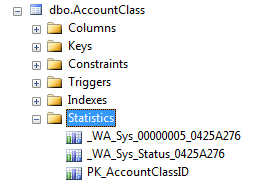
### 相关的子查询

# 统计信息

## 更新统计信息

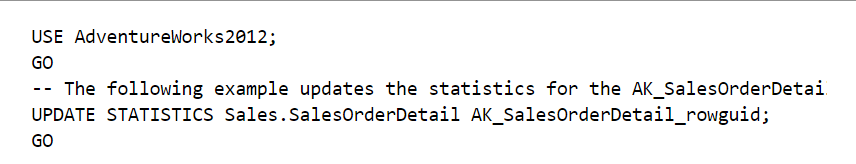
<https://msdn.microsoft.com/zh-cn/library/hh510198.aspx>

### 使用Visual Studio

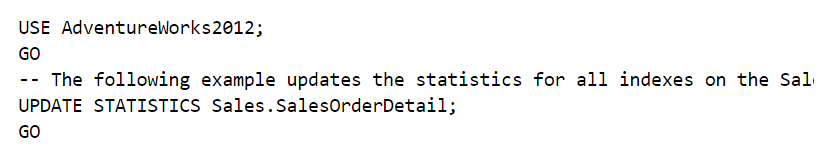


### 使用SQL

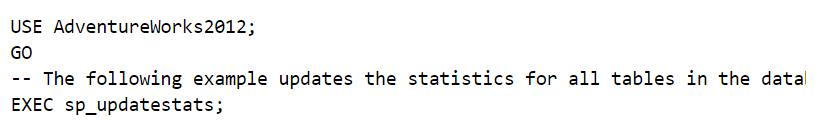
更新特定的统计信息



更新整个表的统计信息



更新所有的统计信息



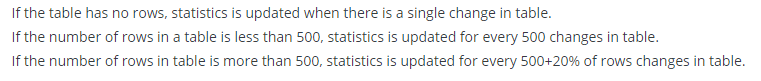
### 慎用sp\_updatestats

会很耗时

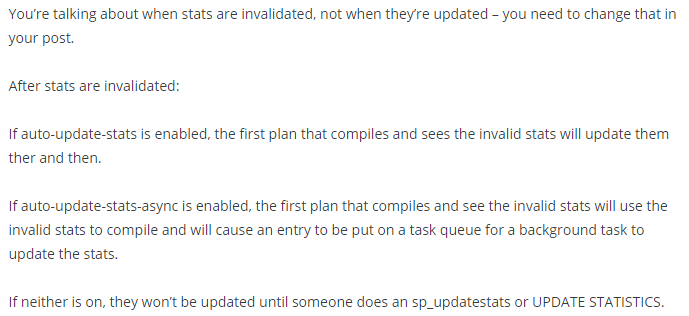
## 统计信息的失效和更新

<http://blog.sqlauthority.com/2010/04/21/sql-server-when-are-statistics-updated-what-triggers-statistics-to-update/>

### 统计信息会失效

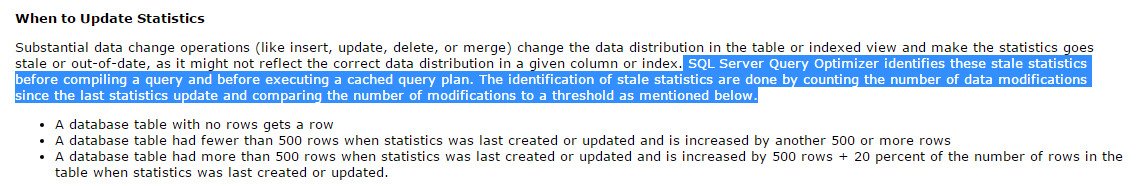


### 统计信息会不会自动更新取决于一些配置



## 查询优化器在执行查询之前会先check 统计信息是否已经失效

[Importance of Statistics and How It Works in SQL Server – Part 1](http://www.databasejournal.com/features/mssql/importance-of-statistics-and-how-it-works-in-sql-server-part-1.html)



统计失效比较的是变化的数据条数，而不仅仅是新添加的数据条数。因为update 某一列的值也属于变化

查询优化器检测到失效后，是不是更新统计信息，还取决于系统的配置：

Auto-update-stats

Auto-update-stats-async

## 有了自动更新，为什么查询优化时还会建议手动更新统计信息

根据统计信息失效的计算公式可以看到，

表中的数据大于500条的时候，但总条数的20% + 500 变化了之后，才触发统计信息失效。

考虑一个1000万条记录的表，20%就是200万条变化

在这个范围内的，都认为是有效的统计信息，即使配置了自动更新，查询优化器也是不会更新统计信息的。

所以为了查询优化，仍建议手动的更新一下统计信息

# 性能调优和查询优化

## Performance Tuning VS Query Optimization

这两个的含义其实是不太一样的。

Performance Tuning and Query Optimization。

查询优化是性能调优的其中一个手段。在不改变query的情况下，添加硬件，更快的CPU，更多的内存， 改变软件，如应用软件访问数据库的形式：提交一个复杂查询还是分步提交多个简单查询，都是性能调优的范围。

而查询优化则近指的从query 角度进行的优化，可以包括

* 更新统计信息
* 改写query的写法（避免游标等，不在索引列上计算）
* 添加索引等
* 修改索引（索引的列不够，或者顺序不合适）

## SQL Server调优系列进阶篇（深入剖析统计信息）

<http://blog.jobbole.com/82845/>

里面的一系列连接值得好好看看

## 注意操作执行的次数

执行一次的往往比较快，但如果执行了成千上万次，那就往往比较耗时了

## 注意返回的记录条数

可以看看图形化的线条的粗细

## 注意TempDB

许多数据都需要借助TempDB 来实现。