[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？ 2](#_Toc465079045)

[1-1. 调用函数时，默认参数不能省 2](#_Toc465079046)

[2. Group by 和Case 语句 3](#_Toc465079047)

[3. Row\_Number（）的使用 3](#_Toc465079048)

[3-1. 返回row number 4](#_Toc465079049)

[3-2. 使用 row number 来返回分页数据 4](#_Toc465079050)

[3-3. 在row number 中使用分组 5](#_Toc465079051)

[4. 使用sql 找出当月的天数 5](#_Toc465079052)

[5. Cast vs Convert 5](#_Toc465079053)

[5-1. Cast 在sql server 内部其实还是通过convert 实现的 6](#_Toc465079054)

[5-2. 转换为char，varchar 时，不指定长度的话，默认是30 7](#_Toc465079055)

[6. SQL 优化 7](#_Toc465079056)

[7. 插入10万条数据大概需要多少时间 7](#_Toc465079057)

[8. 查询sql server 的版本 8](#_Toc465079058)

[8-1. 14的版本 8](#_Toc465079059)

[8-2. 12的版本 8](#_Toc465079060)

[8-3. 2008的版本 8](#_Toc465079061)

[8-4. 2005的版本 9](#_Toc465079062)

[8-5. 2000的版本 9](#_Toc465079063)

[9. 索引和数据的存储 9](#_Toc465079064)

# 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。

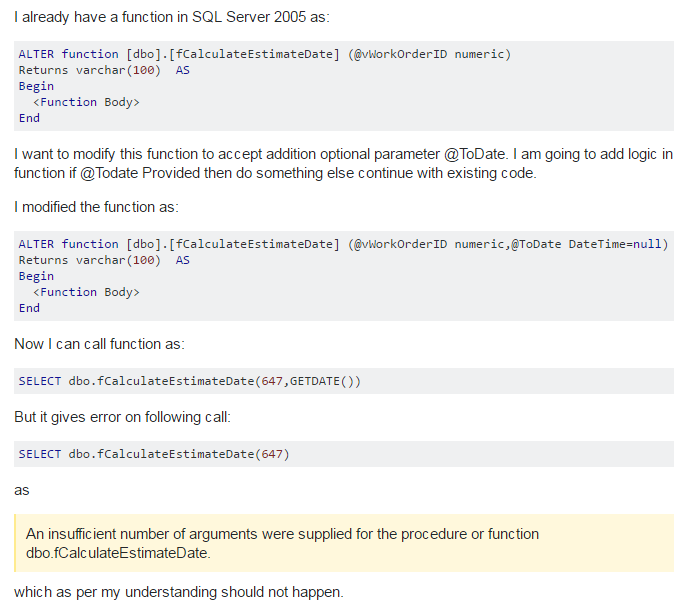
存储过程，其返回值不能被直接引用。 自定义函数，其返回值可以被直接引用。

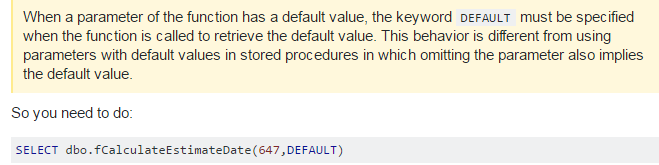
存储过程，可返回记录集。 自定义函数，可以返回表变量。

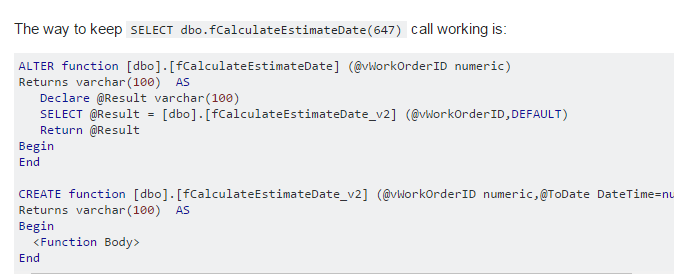
## 调用函数时，默认参数不能省

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

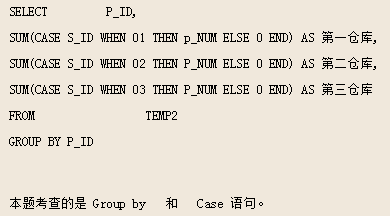
但是在执行的时候，旧有逻辑报错，说函数的参数数目不对，一查原来是调用的时候不能省啊，必须传入default才行。 要不就加一个新函数，在旧函数里默认调用这个新函数







# Group by 和Case 语句

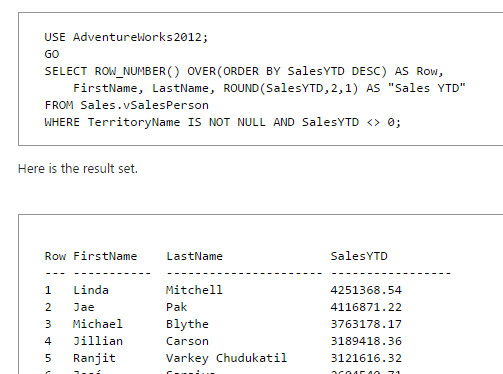


# Row\_Number（）的使用

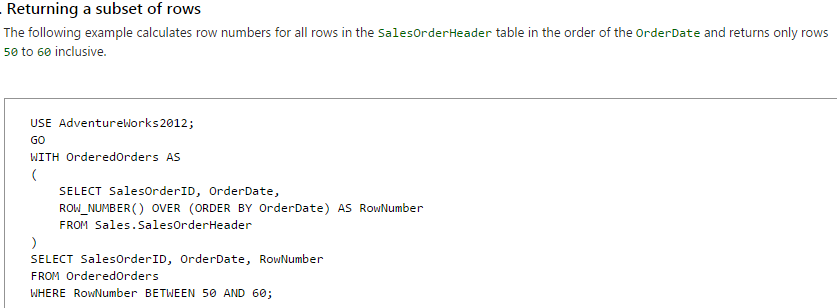
Row\_number() 必须和over 一起使用， over 里面必须至少有一个order by 语句，可以可选一个分组语句



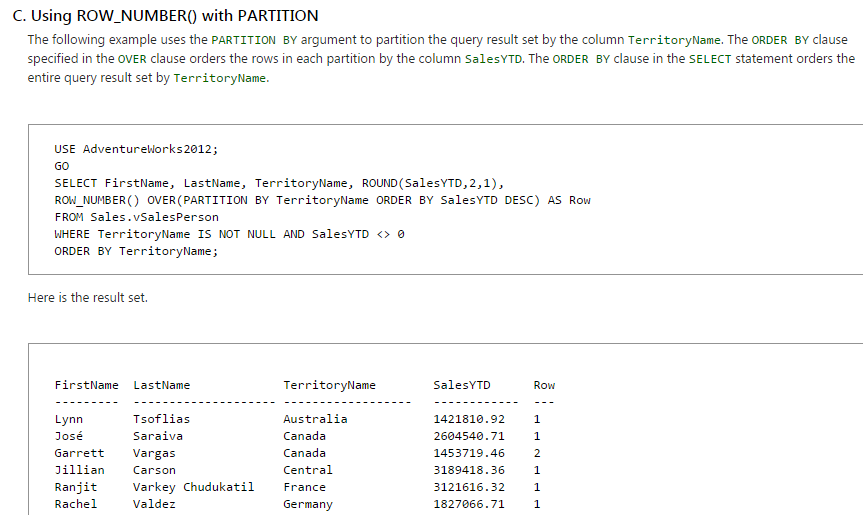
## 返回row number



## 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

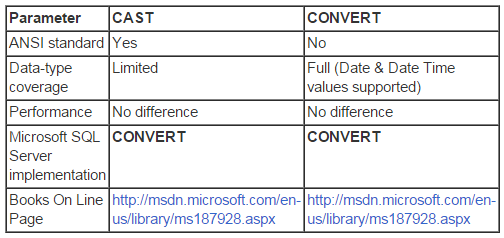
用dateadd(dd,-1) 转换为前月最后一天

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



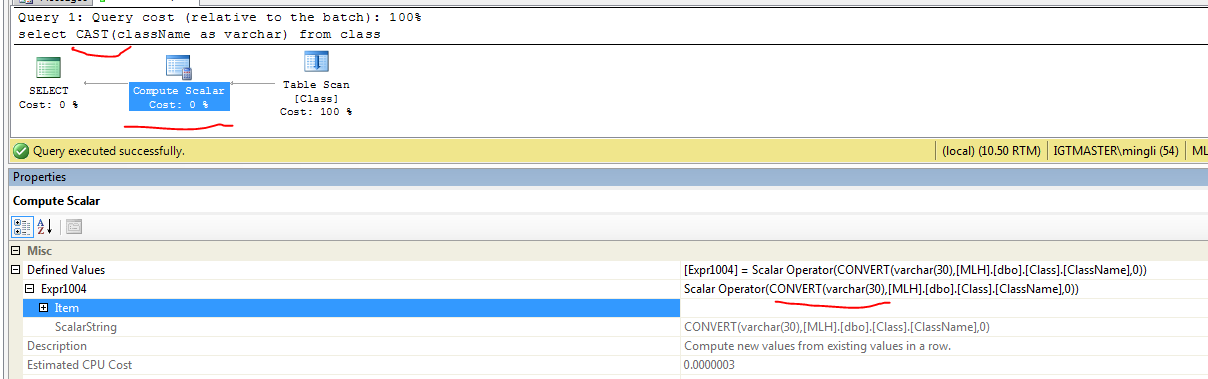
CONVERT is specific to SQL Server, and allows for a greater breadth of flexibility when converting between date and time values, fractional numbers, and monetary signifiers.

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.

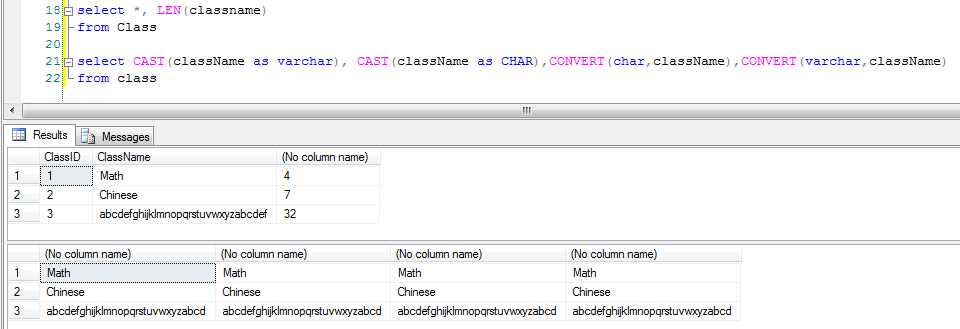
有时候结合这两个一起使用会更灵活。

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

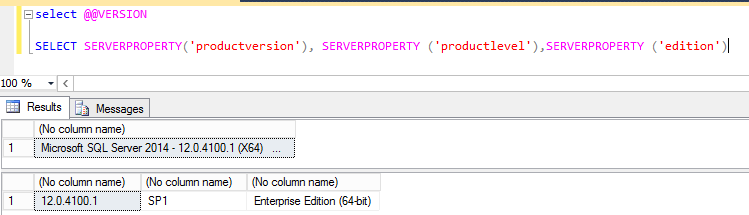
# 查询sql server 的版本

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

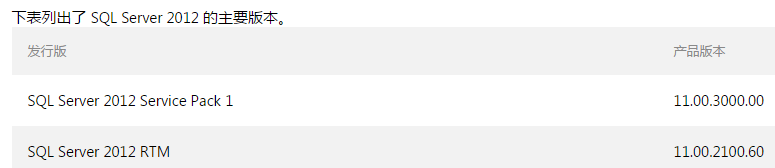




## 14的版本



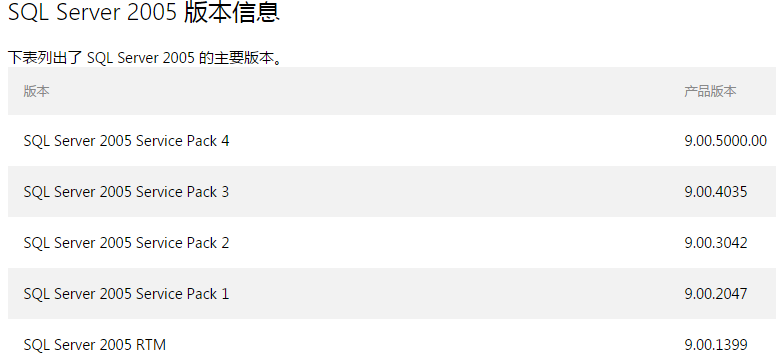
## 12的版本



## 2008的版本



## 2005的版本



## 2000的版本



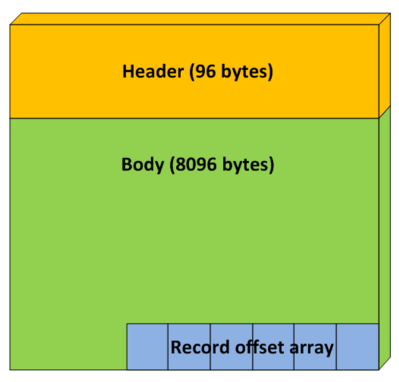
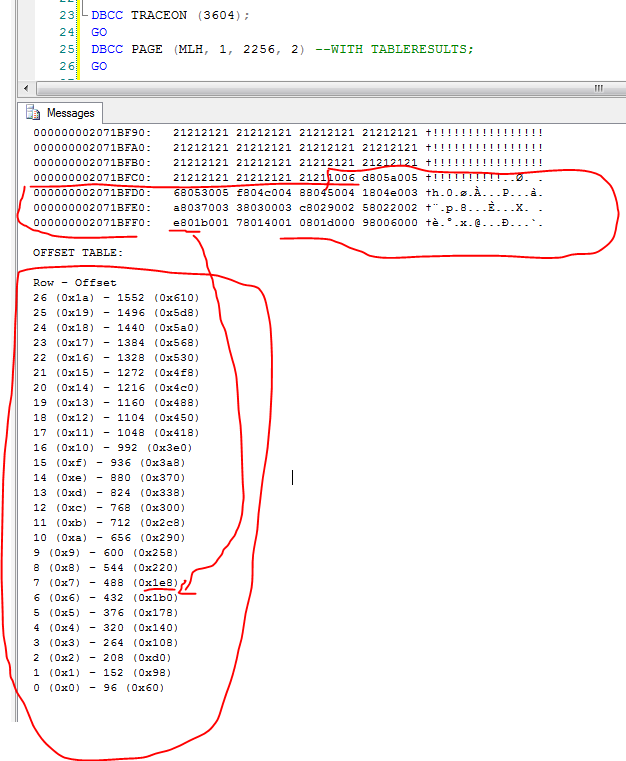
# 索引和数据的存储

[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

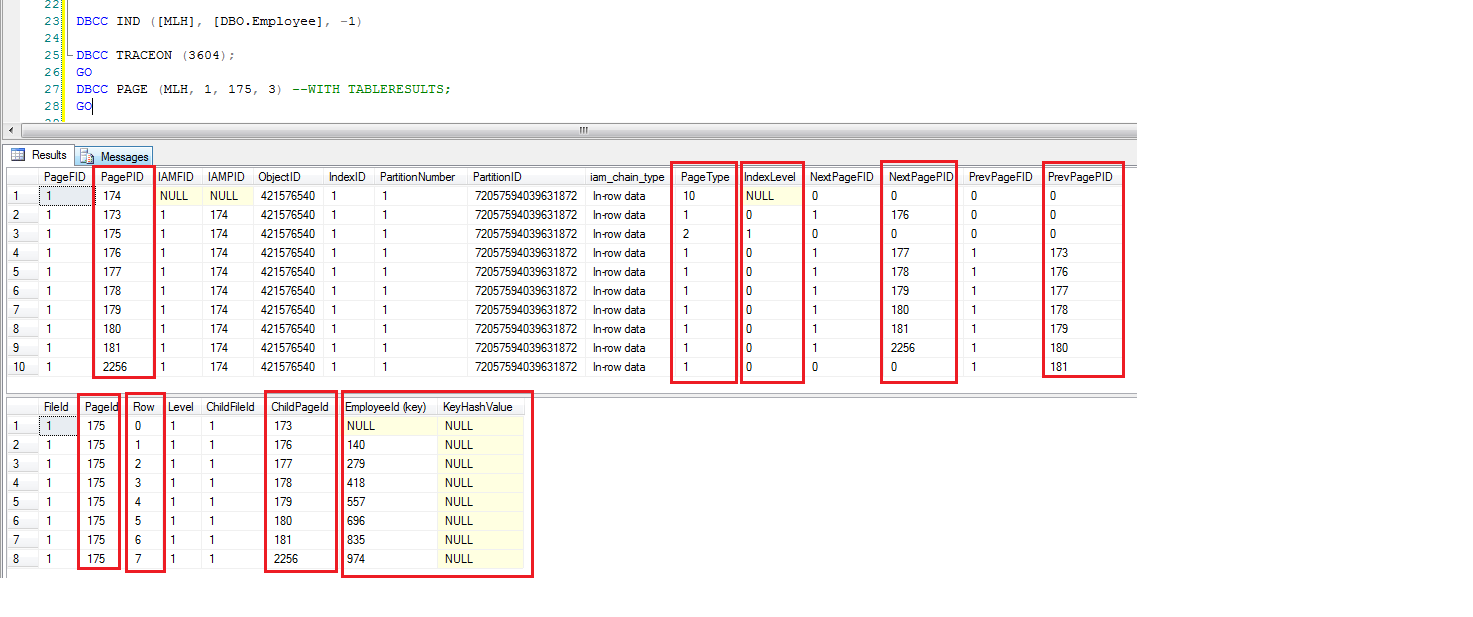
1. 不管是数据页还是索引列， 里面的数据或者索引都是以行结构来存储的。对于数据页来说，每一行就是普通的一行数据，里面包含各个列的数据。对于索引行来说，每一行的数据包括索引的字段（联合索引的话应该会存索引中的所有的字段），（主键信息字段 待定，或者是对应行的hash 信息）， 以及指向下一级页节点的指针。
2. 利用 DBCC IND 和 DBCC PAGE 可以构建出 整个B+数，上面是索引节点，下面是叶节点。对于聚集索引来说，叶节点就是物理的数据。对于非聚集索引来说，页节点是指向为例数据的行标签。

[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。