数据量过大时数据库操作的处理

随着"金盾工程"建设的逐步深入和公安信息化的高速发展，公安计算机应用系统被广泛应用在各警种、各部门。与此同时，应用系统体系的核心、系统数据的存放地——数据库也随着实际应用而急剧膨胀，一些大规模的系统，如人口系统的数据甚至超过了 1000 万条，可谓海量。那么，如何实现快速地从这些超大容量的数据库中提取数据(查询)、分析、统计以及提取数据后进行数据分页已成为各地系统管理员和数据库管理员亟待解决的难题。

在以下的文章中，我将以"办公自动化"系统为例，探讨如何在有着 1000 万条数据的 MS SQL SERVER 数据库中实现快速的数据提取和数据分页。以下代码说明了我们实例中数据库的"红头文件"一表的部分数据结构:

```
CREATE TABLE [dbo].[TGongwen] (     --TGongwen 是红头文件表名
   [Gid] [int] IDENTITY (1, 1) NOT NULL ,
--本表的 id 号，也是主键
   [title] [varchar] (80) COLLATE Chinese_PRC_CI_AS NULL ,
--红头文件的标题
   [fariqi] [datetime] NULL ,
--发布日期
   [neibuYonghu] [varchar] (70) COLLATE Chinese_PRC_CI_AS NULL ,
--发布用户
   [reader] [varchar] (900) COLLATE Chinese_PRC_CI_AS NULL ,
--需要浏览的用户。每个用户中间用分隔符",”分开
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

下面，我们来往数据库中添加 1000 万条数据:

```
declare @i int
set @i=1
while @i<=250000
begin
    insert into Tgongwen(fariqi,neibuyonghu,reader,title) values('2004-2-5','通信科','通信科,办公室,王局长,刘局长,张局长,admin,刑侦支队,特勤支队,交巡警支队,经侦支队,户政科,治安支队,外事科','这是最先的 25 万条记录')
    set @i=@i+1
end
GO

declare @i int
set @i=1
while @i<=250000
```

```
begin
    insert into Tgongwen(fariqi,neibuyonghu,reader,title) values('2004-9-16','办公室','办公室,通信科,王局长,刘局长,张局长,admin,刑侦支队,特勤支队,交巡警支队,经侦支队,户政科,外事科','这是中间的 25 万条记录')
    set @i=@i+1
end
GO

declare @h int
set @h=1
while @h<=100
begin
declare @i int
set @i=2002
while @i<=2003
begin
declare @j int
        set @j=0
        while @j<50
            begin
declare @k int
            set @k=0
            while @k<50
            begin
    insert into Tgongwen(fariqi,neibuyonghu,reader,title) values(cast(@i as varchar(4))+'-8-15 3:'+cast(@j as varchar(2))+':'+cast(@j as varchar(2)),'通信科','办公室,通信科,王局长,刘局长,张局长,admin,刑侦支队,特勤支队,交巡警支队,经侦支队,户政科,外事科','这是最后的 50 万条记录')
            set @k=@k+1
            end
set @j=@j+1
        end
set @i=@i+1
end
set @h=@h+1
end
GO

declare @i int
set @i=1
while @i<=9000000
begin
    insert into Tgongwen(fariqi,neibuyonghu,reader,title) values('2004-5-5','通信科','通信科,办公室,王局长,刘局长,张局长,admin,刑侦支队,特勤支队,交巡警支队,经侦支队,户政科,治安支
```

```
队,外事科','这是最后添加的 900 万条记录')
    set @i=@i+1000000
end
GO
```

通过以上语句，我们创建了 25 万条由通信科于 2004 年 2 月 5 日发布的记录，25 万条由办公室于 2004 年 9 月 6 日发布的记录，2002 年和 2003 年各 100 个 2500 条相同日期、不同分秒的由通信科发布的记录(共 50 万条)，还有由通信科于 2004 年 5 月 5 日发布的 900 万条记录，合计 1000 万条。

一、因情制宜，建立"适当"的索引

建立"适当"的索引是实现查询优化的首要前提。

索引(index)是除表之外另一重要的、用户定义的存储在物理介质上的数据结构。当根据索引码的值搜索数据时，索引提供了对数据的快速访问。事实上，没有索引,数据库也能根据 SELECT 语句成功地检索到结果，但随着表变得越来越大，使用"适当"的索引的效果就越来越明显。注意，在这句话中，我们用了"适当"这个词，这是因为，如果使用索引时不认真考虑其实现过程，索引既可以提高也会破坏数据库的工作性能。

(一)深入浅出理解索引结构

实际上，您可以把索引理解为一种特殊的目录。微软的 SQL SERVER 提供了两种索引:聚集索引(clustered index，也称聚类索引、簇集索引)和非聚集索引(nonclustered index，也称非聚类索引、非簇集索引)。下面，我们举例来说明一下聚集索引和非聚集索引的区别:

其实，我们的汉语字典的正文本身就是一个聚集索引。比如，我们要查"安"字，就会很自然地翻开字典的前几页，因为"安"的拼音是"an"，而按照拼音排序汉字的字典是以英文字母"a"开头并以"z"结尾的，那么"安"字就自然地排在字典的前部。如果您翻完了所有以"a"开头的部分仍然找不到这个字，那么就说明您的字典中没有这个字;同样的，如果查"张"字，那您也会将您的字典翻到最后部分，因为"张"的拼音是"zhang"。也就是说，字典的正文部分本身就是一个目录，您不需要再去查其他目录来找到您需要找的内容。

我们把这种正文内容本身就是一种按照一定规则排列的目录称为"聚集索引"。

如果您认识某个字，您可以快速地从自动中查到这个字。但您也可能会遇到您不认识的字，不知道它的发音，这时候，您就不能按照刚才的方法找到您要查的字，而需要去根据"偏旁部首"查到您要找的字，然后根据这个字后的页码直接翻到某页来找到您要找的字。但您结合"部首目录"和"检字表"而查到的字的排序并不是真正的正文的排序方法，比如您查"张"字，我们可以看到在查部首之后的检字表中"张"的页码是 672 页，检字表中"张"的上面是"驰"字，但页码却是 63 页，"张"的下面是"弩"字，页面是 390 页。很显然，这些字并不是真正的分别位于"张"字的上下方，现在您看到的连续的"驰、张、弩"三字实际上就是他们在非聚集索引中的排序，是字典正文中的字在非聚集索引中的映射。我们可以通过这种方式来找到您所需要的字，但它需要两个过程，先找到目录中的结果，然后再翻到您所需要的页码。

我们把这种目录纯粹是目录，正文纯粹是正文的排序方式称为"非聚集索引"。

通过以上例子，我们可以理解到什么是"聚集索引"和"非聚集索引"。

进一步引申一下，我们可以很容易的理解:每个表只能有一个聚集索引，因为目录只能按照一种方法进行排序。

(二)何时使用聚集索引或非聚集索引

下面的表总结了何时使用聚集索引或非聚集索引(很重要)。

动作描述	使用聚集索引	使用非聚集索引
列经常被分组排序	应	应
返回某范围内的数据	应	不应
一个或极少不同值	不应	不应
小数目的不同值	应	不应
大数目的不同值	不应	应
频繁更新的列	不应	应
外键列	应	应
主键列	应	应
频繁修改索引列	不应	应

事实上，我们可以通过前面聚集索引和非聚集索引的定义的例子来理解上表。如:返回某范围内的数据一项。比如您的某个表有一个时间列，恰好您把聚合索引建立在了该列，这时您查询 2004 年 1 月 1 日至 2004 年 10 月 1 日之间的全部数据时，这个速度就将是很快的，因为您的这本字典正文是按日期进行排序的，聚类索引只需要找到要检索的所有数据中的开头和结尾数据即可;而不像非聚集索引，必须先查到目录中查到每一项数据对应的页码，然后再根据页码查到具体内容。

(三)结合实际，谈索引使用的误区

理论的目的是应用。虽然我们刚才列出了何时应使用聚集索引或非聚集索引，但在实践中以上规则却很容易被忽视或不能根据实际情况进行综合分析。下面我们将根据在实践中遇到的实际问题来谈一下索引使用的误区，以便于大家掌握索引建立的方法。

1、主键就是聚集索引

这种想法笔者认为是极端错误的，是对聚集索引的一种浪费。虽然 SQL SERVER 默认是在主键上建立聚集索引的。

通常，我们会在每个表中都建立一个 ID 列，以区分每条数据，并且这个 ID 列是自动增大的，步长一般为 1。我们的这个办公自动化的实例中的列 Gid 就是如此。此时，如果我们将这个列设为主键，SQL SERVER 会将此列默认为聚集索引。这样做有好处，就是可以让您的数据在数据库中按照 ID 进行物理排序，但笔者认为这样做意义不大。

显而易见，聚集索引的优势是很明显的，而每个表中只能有一个聚集索引的规则，这使得聚集索引变得更加珍贵。

从我们前面谈到的聚集索引的定义我们可以看出，使用聚集索引的最大好处就是能够根据查询要求，迅速缩小查询范围，避免全表扫描。在实际应用中，因为 ID 号是自动生成的，我们并不知道每条记录的 ID 号，所以我们很难在实践中用 ID 号来进行查询。这就使让 ID 号这个主键作为聚集索引成为一种资源浪费。其次，让每个 ID 号都不同的字段作为聚集索引也不符合"大数目的不同值情况下不应建立聚合索引"规则；当然，这种情况只是针对用户经常修改记录内容，特别是索引项的时候会负作用，但对于查询速度并没有影响。

在办公自动化系统中，无论是系统首页显示的需要用户签收的文件、会议还是用户进行文件查询等任何情况下进行数据查询都离不开字段的是"日期"还有用户本身的"用户名"。

通常，办公自动化的首页会显示每个用户尚未签收的文件或会议。虽然我们的 where 语句可以仅仅限制当前用户尚未签收的情况，但如果您的系统已建立了很长时间，并且数据量很大，那么，每次每个用户打开首页的时候都进行一次全表扫描，这样做意义是不大的，绝大多数的用户 1 个月前的文件都已经浏览过了，这样做只能徒增数据库的开销而已。事实上，我们完全可以让用户打开系统首页时，数据库仅仅查询这个用户近 3 个月来未阅览的文件，通过"日期"这个字段来限制表扫描，提高查询速度。如果您的办公自动化系统已经建立的 2 年，那么您的首页显示速度理论上将是原来速度 8 倍，甚至更快。

在这里之所以提到"理论上"三字，是因为如果您的聚集索引还是盲目地建在 ID 这个主键上时，您的查询速度是没有这么高的，即使您在"日期"这个字段上建立的索引(非聚合索引)。下面我们就来看一下在 1000 万条数据量的情况下各种查询的速度表现(3 个月内的数据为 25 万条):

(1)仅在主键上建立聚集索引，并且不划分时间段:

```
Select gid,fariqi,neibuyonghu,title from tgongwen
```

用时:128470 毫秒(即:128 秒)

(2)在主键上建立聚集索引，在 fariq 上建立非聚集索引:

```
select gid,fariqi,neibuyonghu,title from Tgongwen
where fariqi> dateadd(day,-90,getdate())
```

用时:53763 毫秒(54 秒)

(3)将聚合索引建立在日期列(fariqi)上:

```
select gid,fariqi,neibuyonghu,title from Tgongwen
where fariqi> dateadd(day,-90,getdate())
```

用时:2423 毫秒(2 秒)

虽然每条语句提取出来的都是 25 万条数据，各种情况的差异却是巨大的，特别是将聚集索引建立在日期列时的差异。事实上，如果您的数据库真的有 1000 万容量的话，把主键建立在 ID 列上，就像以上的第 1、2 种情况，在网页上的表现就是超时，根本就无法显示。这也是我摒弃 ID 列作为聚集索引的一个最重要的因素。

得出以上速度的方法是:在各个 select 语句前加:

```
declare @d datetime
set @d=getdate()
```

并在 select 语句后加:

```
select [语句执行花费时间(毫秒)]=datediff(ms,@d,getdate())
```

2、只要建立索引就能显著提高查询速度

事实上，我们可以发现上面的例子中，第 2、3 条语句完全相同，且建立索引的字段也相同;不同的仅是前者在 fariqi 字段上建立的是非聚合索引，后者在此字段上建立的是聚合索引，但查询速度却有着天壤之别。所以，并非是在任何字段上简单地建立索引就能提高查询速度。

从建表的语句中，我们可以看到这个有着 1000 万数据的表中 fariqi 字段有 5003 个不同记录。在此字段上建立聚合索引是再合适不过了。在现实中，我们每天都会发几个文件，这几个文件的发文日期就相同，这完全符合建立聚集索引要求的:“既不能绝大多数都相同，又不能只有极少数相同”的规则。由此看来，我们建立“适当”的聚合索引对于我们提高查询速度是非常重要的。

3、把所有需要提高查询速度的字段都加进聚集索引，以提高查询速度

上面已经谈到:在进行数据查询时都离不开字段的是“日期”还有用户本身的“用户名”。既然这两个字段都是如此的重要，我们可以把他们合并起来，建立一个复合索引(compound index)。

很多人认为只要把任何字段加进聚集索引，就能提高查询速度，也有人感到迷惑:如果把复合的聚集索引字段分开查询，那么查询速度会减慢吗?带着这个问题，我们来看一下以下的查询速度(结果集都是 25 万条数据):(日期列 fariqi 首先排在复合聚集索引的起始列，用户名 neibuyonghu 排在后列)

(1)select gid,fariqi,neibuyonghu,title from Tgongwen where fariqi>'2004-5-5'

查询速度:2513 毫秒

(2)select gid,fariqi,neibuyonghu,title from Tgongwen where fariqi>'2004-5-5' and neibuyonghu='办公室'

查询速度:2516 毫秒

(3)select gid,fariqi,neibuyonghu,title from Tgongwen where neibuyonghu='办公室'

查询速度:60280 毫秒

从以上试验中，我们可以看到如果仅用聚集索引的起始列作为查询条件和同时用到复合聚集索引的全部列的查询速度是几乎一样的，甚至比用上全部的复合索引列还要略快(在查询结果集数目一样的情况下);而如果仅用复合聚集索引的非起始列作为查询条件的话，这个索引是不起任何作用的。当然，语句 1、2 的查询速度一样是因为查询的条目数一样，如果复合索引的所有列都用上，而且查询结果少的话，这样就会形成“索引覆盖”，因而性能可以达到最优。同时，请记住:无论您是否经常使用聚合索引的其他列，但其前导列一定要是使用最频繁的列。

(四)其他书上没有的索引使用经验总结

1、用聚合索引比用不是聚合索引的主键速度快

下面是实例语句:(都是提取 25 万条数据)

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where fariqi='2004-9-16'
```

使用时间:3326 毫秒

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where gid<=250000
```

使用时间:4470 毫秒

这里，用聚合索引比用不是聚合索引的主键速度快了近 1/4。

2、用聚合索引比用一般的主键作 order by 时速度快，特别是在小数据量情况下

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen order by fariqi
```

用时:12936

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen order by gid
```

这里，用聚合索引比用一般的主键作 order by 时，速度快了 3/10。事实上，如果数据量很小的话，用聚集索引作为排序列要比使用非聚集索引速度快得明显的多;而数据量如果很大的话，如 10 万以上，则二者的速度差别不明显。

3、使用聚合索引内的时间段，搜索时间会按数据占整个数据表的百分比成比例减少，而无论聚合索引使用了多少个

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where fariqi>'2004-1-1'
```

用时:6343 毫秒(提取 100 万条)

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where fariqi>'2004-6-6'
```

用时:3170 毫秒(提取 50 万条)

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where fariqi='2004-9-16'
```

用时:3326 毫秒(和上句的结果一模一样。如果采集的数量一样，那么用大于号和等于号是一样的)

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where fariqi>'2004-1-1' and fariqi<'2004-6-6'
```

用时:3280 毫秒

4 、日期列不会因为有分秒的输入而减慢查询速度

下面的例子中，共有 100 万条数据，2004 年 1 月 1 日以后的数据有 50 万条，但只有两个不同的日期，日期精确到日;之前有数据 50 万条，有 5000 个不同的日期，日期精确到秒。

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where fariqi>'2004-1-1' order by fariqi
```

用时:6390 毫秒

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where fariqi<'2004-1-1' order by fariqi
```

用时:6453 毫秒

(五)其他注意事项

“水可载舟，亦可覆舟”，索引也一样。索引有助于提高检索性能，但过多或不当的索引也会导致系统低效。因为用户在表中每加进一个索引，数据库就要做更多的工作。过多的索引甚至会导致索引碎片。

所以说，我们要建立一个“适当”的索引体系，特别是对聚合索引的创建，更应精益求精，以使您的数据库能得到高性能的发挥。

当然，在实践中，作为一个尽职的数据库管理员，您还要多测试一些方案，找出哪种方案效率最高、最为有效。

二、改善 SQL 语句

很多人不知道 SQL 语句在 SQL SERVER 中是如何执行的，他们担心自己所写的 SQL 语句会被 SQL SERVER 误解。比如:

```
select * from table1 where name='zhangsan' and tID > 10000
```

和执行:

```
select * from table1 where tID > 10000 and name='zhangsan'
```

一些人不知道以上两条语句的执行效率是否一样，因为如果简单的从语句先后上看，这两个语句的确是不一样，如果 tID 是一个聚合索引，那么后一句仅仅从表的 10000 条以后的记录中查找就行了;而前一句则要先从全表中查找看有几个 name='zhangsan'的，而后再根据限制条件条件 tID>10000 来提出查询结果。

事实上，这样的担心是不必要的。SQL SERVER 中有一个“查询分析优化器”，它可以计算出 where 子句中的搜索条件并确定哪个索引能缩小表扫描的搜索空间，也就是说，它能实现自动优化。

虽然查询优化器可以根据 where 子句自动的进行查询优化，但大家仍然有必要了解一下“查询优化器”的工作原理，如非这样，有时查询优化器就会不按照您的本意进行快速查询。

在查询分析阶段，查询优化器查看查询的每个阶段并决定限制需要扫描的数据量是否有用。如果一个阶段可以被用作一个扫描参数(SARG)，那么就称之为可优化的，并且可以利用索引快速获得所需数据。

SARG 的定义:用于限制搜索的一个操作，因为它通常是指一个特定的匹配，一个值得范围内的匹配或者两个以上条件的 AND 连接。形式如下:

列名 操作符 <常数 或 变量>

或

<常数 或 变量> 操作符列名

列名可以出现在操作符的一边，而常数或变量出现在操作符的另一边。如:

Name='张三'

价格>5000

5000<价格

Name='张三' and 价格>5000

如果一个表达式不能满足 SARG 的形式，那它就无法限制搜索的范围了，也就是 SQL SERVER 必须对每一行都判断它是否满足 WHERE 子句中的所有条件。所以一个索引对于不满足 SARG 形式的表达式来说是无用的。

介绍完 SARG 后，我们来总结一下使用 SARG 以及在实践中遇到的和某些资料上结论不同的经验:

1、Like 语句是否属于 SARG 取决于所使用的通配符的类型

如:name like '张%' ，这就属于 SARG

而:name like '%张' ,就不属于 SARG。

原因是通配符%在字符串的开通使得索引无法使用。

2、or 会引起全表扫描

Name='张三' and 价格>5000 符号 SARG，而:Name='张三' or 价格>5000 则不符合 SARG。使用 or 会引起全表扫描。

3、非操作符、函数引起的不满足 SARG 形式的语句

不满足 SARG 形式的语句最典型的情况就是包括非操作符的语句，如:NOT、!=、<>、!<、!>、NOT EXISTS、NOT IN、NOT LIKE 等，另外还有函数。下面就是几个不满足 SARG 形式的例子:

ABS(价格)<5000

Name like '%三'

有些表达式，如:

WHERE 价格*2>5000

SQL SERVER 也会认为是 SARG，SQL SERVER 会将此式转化为:

WHERE 价格>2500/2

但我们不推荐这样使用，因为有时 SQL SERVER 不能保证这种转化与原始表达式是完全等价的。

4、IN 的作用相当与 OR

语句:

```
Select * from table1 where tid in (2,3)
```

和

```
Select * from table1 where tid=2 or tid=3
```

是一样的，都会引起全表扫描，如果 tid 上有索引，其索引也会失效。

5、尽量少用 NOT

6、exists 和 in 的执行效率是一样的

很多资料上都显示说，exists 要比 in 的执行效率要高，同时应尽可能的用 not exists 来代替 not in。但事实上，我试验了一下，发现二者无论是前面带不带 not，二者之间的执行效率都是一样的。因为涉及子查询，我们试验这次用 SQL SERVER 自带的 pubs 数据库。运行前我们可以把 SQL SERVER 的 statistics I/O 状态打开。

(1)select title,price from titles where title_id in (select title_id from sales where qty>30)

该句的执行结果为:

表 'sales'。扫描计数 18，逻辑读 56 次，物理读 0 次，预读 0 次。

表 'titles'。扫描计数 1，逻辑读 2 次，物理读 0 次，预读 0 次。

(2)select title,price from titles where exists (select * from sales where sales.title_id=titles.title_id and qty>30)

第二句的执行结果为:

表 'sales'。扫描计数 18，逻辑读 56 次，物理读 0 次，预读 0 次。

表 'titles'。扫描计数 1，逻辑读 2 次，物理读 0 次，预读 0 次。

我们从此可以看到用 exists 和用 in 的执行效率是一样的。

7、用函数 charindex()和前面加通配符%的 LIKE 执行效率一样

前面，我们谈到，如果在 LIKE 前面加上通配符%，那么将会引起全表扫描，所以其执行效率是低下的。但有的资料介绍说，用函数 charindex()来代替 LIKE 速度会有大的提升，经我试验，发现这种说明也是错误的:

```
select gid,title,fariqi,reader from tgongwen where charindex('刑侦支队',reader)>0 and fariqi>'2004-5-5'
```

用时:7 秒，另外:扫描计数 4，逻辑读 7155 次，物理读 0 次，预读 0 次。

```
select gid,title,fariqi,reader from tgongwen where reader like '%' + '刑侦支队' + '%' and fariqi>'2004-5-5'
```

用时:7 秒，另外:扫描计数 4，逻辑读 7155 次，物理读 0 次，预读 0 次。

8、union 并不绝对比 or 的执行效率高

我们前面已经谈到了在 where 子句中使用 or 会引起全表扫描，一般的，我所见过的资料都是推荐这里用 union 来代替 or。事实证明，这种说法对于大部分都是适用的。

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where fariqi='2004-9-16' or gid>9990000
```

用时:68 秒。扫描计数 1，逻辑读 404008 次，物理读 283 次，预读 392163 次。

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where fariqi='2004-9-16'
union
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where gid>9990000
```

用时:9 秒。扫描计数 8，逻辑读 67489 次，物理读 216 次，预读 7499 次。

看来，用 union 在通常情况下比用 or 的效率要高的多。

但经过试验，笔者发现如果 or 两边的查询列是一样的话，那么用 union 则反倒和用 or 的执行速度差很多，虽然这里 union 扫描的是索引，而 or 扫描的是全表。

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where fariqi='2004-9-16' or fariqi='2004-2-5'
```

用时:6423 毫秒。扫描计数 2，逻辑读 14726 次，物理读 1 次，预读 7176 次。

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where fariqi='2004-9-16'
union
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where fariqi='2004-2-5'
```

用时:11640 毫秒。扫描计数 8，逻辑读 14806 次，物理读 108 次，预读 1144 次。

9、字段提取要按照"需多少、提多少"的原则，避免"select *"

我们来做一个试验:

```
select top 10000 gid,fariqi,reader,title from tgongwen order by gid desc
```

用时:4673 毫秒

```
select top 10000 gid,fariqi,title from tgongwen order by gid desc
```

用时:1376 毫秒

```
select top 10000 gid,fariqi from tgongwen order by gid desc
```

用时:80 毫秒

由此看来，我们每少提取一个字段，数据的提取速度就会有相应的提升。提升的速度还要看您舍弃的字段的大小来判断。

10、count(*)不比 count(字段)慢

某些资料上说:用*会统计所有列，显然要比一个世界的列名效率低。这种说法其实是没有根据的。我们来看:

```
select count(*) from Tgongwen
```

用时:1500 毫秒

```
select count(gid) from Tgongwen
```

用时:1483 毫秒

```
select count(fariqi) from Tgongwen
```

用时:3140 毫秒

```
select count(title) from Tgongwen
```

用时:52050 毫秒

从以上可以看出，如果用 count(*)和用 count(主键)的速度是相当的，而 count(*)却比其他任何除主键以外的字段汇总速度要快，而且字段越长，汇总的速度就越慢。我想，如果用 count(*)， SQL SERVER 可能会自动查找最小字段来汇总的。当然，如果您直接写 count(主键)将会来的更直接些。

11、order by 按聚集索引列排序效率最高

我们来看:(gid 是主键，fariqi 是聚合索引列)

```
select top 10000 gid,fariqi,reader,title from tgongwen
```

用时:196 毫秒。 扫描计数 1，逻辑读 289 次，物理读 1 次，预读 1527 次。

```
select top 10000 gid,fariqi,reader,title from tgongwen order by gid asc
```

用时:4720 毫秒。 扫描计数 1，逻辑读 41956 次，物理读 0 次，预读 1287 次。

```
select top 10000 gid,fariqi,reader,title from tgongwen order by gid desc
```

用时:4736 毫秒。 扫描计数 1，逻辑读 55350 次，物理读 10 次，预读 775 次。

```
select top 10000 gid,fariqi,reader,title from tgongwen order by fariqi asc
```

用时:173 毫秒。 扫描计数 1，逻辑读 290 次，物理读 0 次，预读 0 次。

```
select top 10000 gid,fariqi,reader,title from tgongwen order by fariqi desc
```

用时:156 毫秒。 扫描计数 1，逻辑读 289 次，物理读 0 次，预读 0 次。

从以上我们可以看出，不排序的速度以及逻辑读次数都是和"order by 聚集索引列" 的速度是相当的，但这些都比"order by 非聚集索引列"的查询速度是快得多的。

同时，按照某个字段进行排序的时候，无论是正序还是倒序，速度是基本相当的。

12、高效的 TOP

事实上，在查询和提取超大容量的数据集时，影响数据库响应时间的最大因素不是数据查找，而是物理的 I/O 操作。如:

```
select top 10 * from (
select top 10000 gid,fariqi,title from tgongwen
where neibuyonghu='办公室'
order by gid desc) as a
order by gid asc
```

这条语句，从理论上讲，整条语句的执行时间应该比子句的执行时间长，但事实相反。因为，子句执行后返回的是 10000 条记录，而整条语句仅返回 10 条语句，所以影响数据库响应时间最大的因素是物理 I/O 操作。而限制物理 I/O 操作此处的最有效方法之一就是使用 TOP 关键词了。TOP 关键词是 SQL SERVER 中经过系统优化过的一个用来提取前几条或前几个百分比数据的词。经笔者在实践中的应用，发现 TOP 确实很好用，效率也很高。但这个词在另外一个大型数据库 ORACLE 中却没有，这不能说不是一个遗憾，虽然在 ORACLE 中可以用其他方法(如:rownumber)来解决。在以后的关于"实现千万级数据的分页显示存储过程"的讨论中，我们就将用到 TOP 这个关键词。

到此为止，我们上面讨论了如何实现从大容量的数据库中快速地查询出您所需要的数据方法。当然，我们介绍的这些方法都是"软"方法，在实践中，我们还要考虑各种"硬"因素，如:网络性能、服务器的性能、操作系统的性能，甚至网卡、交换机等。

三、实现小数据量和海量数据的通用分页显示存储过程

建立一个 web 应用，分页浏览功能必不可少。这个问题是数据库处理中十分常见的问题。经典的数据分页方法是:ADO 纪录集分页法，也就是利用 ADO 自带的分页功能(利用游标)来实现分页。但这种分页方法仅适用于较小数据量的情形，因为游标本身有缺点:游标是存放在内存中，很费内存。游标一建立，就将相关的记录锁住，直到取消游标。游标提供了对特定集合中逐行扫描的手段，一般使用游标来逐行遍历数

据，根据取出数据条件的不同进行不同的操作。而对于多表和大表中定义的游标(大的数据集合)循环很容易使程序进入一个漫长的等待甚至死机。

更重要的是，对于非常大的数据模型而言，分页检索时，如果按照传统的每次都加载整个数据源的方法是非常浪费资源的。现在流行的分页方法一般是检索页面大小的块区的数据，而非检索所有的数据，然后单步执行当前行。

最早较好地实现这种根据页面大小和页码来提取数据的方法大概就是"俄罗斯存储过程"。这个存储过程用了游标，由于游标的局限性，所以这个方法并没有得到大家的普遍认可。

后来，网上有人改造了此存储过程，下面的存储过程就是结合我们的办公自动化实例写的分页存储过程:

```
CREATE procedure pagination1
(@pagesize int,  --页面大小，如每页存储 20 条记录
@pageindex int   --当前页码
)
as
set nocount on
begin
declare @indextable table(id int identity(1,1),nid int)  --定义表变量
declare @PageLowerBound int  --定义此页的底码
declare @PageUpperBound int  --定义此页的顶码
set @PageLowerBound=(@pageindex-1)*@pagesize
set @PageUpperBound=@PageLowerBound+@pagesize
set rowcount @PageUpperBound
insert into @indextable(nid) select gid from TGongwen where
fariqi >dateadd(day,-365,getdate()) order by fariqi desc
select O.gid,O.mid,O.title,O.fadanwei,O.fariqi from TGongwen O,@indextable t where
O.gid=t.nid
and t.id>@PageLowerBound and t.id<=@PageUpperBound order by t.id
end
set nocount off
```

以上存储过程运用了 SQL SERVER 的最新技术——表变量。应该说这个存储过程也是一个非常优秀的分页存储过程。当然，在这个过程中，您也可以把其中的表变量写成临时表:CREATE TABLE #Temp。但很明显，在 SQL SERVER 中，用临时表是没有用表变量快的。所以笔者刚开始使用这个存储过程时，感觉非常的不错，速度也比原来的 ADO 的好。但后来，我又发现了比此方法更好的方法。

笔者曾在网上看到了一篇小短文《从数据表中取出第 n 条到第 m 条的记录的方法》，全文如下:

从 publish 表中取出第 n 条到第 m 条的记录:

```
SELECT TOP m-n+1 *
FROM publish
WHERE (id NOT IN
```

```
(SELECT TOP n-1 id
FROM publish))
```

id 为 publish 表的关键字

我当时看到这篇文章的时候，真的是精神为之一振，觉得思路非常得好。等到后来，我在作办公自动化系统(ASP.NET+ C#+SQL SERVER)的时候，忽然想起了这篇文章，我想如果把这个语句改造一下，这就可能是一个非常好的分页存储过程。于是我就满网上找这篇文章，没想到，文章还没找到，却找到了一篇根据此语句写的一个分页存储过程，这个存储过程也是目前较为流行的一种分页存储过程，我很后悔没有争先把这段文字改造成存储过程:

```
CREATE PROCEDURE pagination2
(
 @SQL nVARCHAR(4000),     --不带排序语句的 SQL 语句
 @Page int,               --页码
 @RecsPerPage int,        --每页容纳的记录数
 @ID VARCHAR(255),        --需要排序的不重复的 ID 号
 @Sort VARCHAR(255)       --排序字段及规则
)
AS

DECLARE @Str nVARCHAR(4000)

SET @Str='SELECT   TOP '+CAST(@RecsPerPage AS VARCHAR(20))+' * FROM
('+@SQL+') T WHERE T.'+@ID+'NOT IN
(SELECT   TOP '+CAST((@RecsPerPage*(@Page-1)) AS VARCHAR(20))+' '+@ID+'
FROM ('+@SQL+') T9 ORDER BY '+@Sort+') ORDER BY '+@Sort

PRINT @Str

EXEC sp_ExecuteSql @Str
GO
```

其实，以上语句可以简化为:

```
SELECT TOP 页大小 *
FROM Table1
WHERE (ID NOT IN
        (SELECT TOP 页大小*页数 id
        FROM 表
        ORDER BY id))
ORDER BY ID
```

但这个存储过程有一个致命的缺点，就是它含有 NOT IN 字样。虽然我可以把它改造为:

```
SELECT TOP 页大小 *
```

```
FROM Table1
WHERE not exists
(select * from (select top (页大小*页数) * from table1 order by id) b where b.id=a.id )
order by id
```

即，用 not exists 来代替 not in，但我们前面已经谈过了，二者的执行效率实际上是没有区别的。

既便如此，用 TOP 结合 NOT IN 的这个方法还是比用游标要来得快一些。

虽然用 not exists 并不能挽救上个存储过程的效率，但使用 SQL SERVER 中的 TOP 关键字却是一个非常明智的选择。因为分页优化的最终目的就是避免产生过大的记录集，而我们在前面也已经提到了 TOP 的优势，通过 TOP 即可实现对数据量的控制。

在分页算法中，影响我们查询速度的关键因素有两点:TOP 和 NOT IN。TOP 可以提高我们的查询速度，而 NOT IN 会减慢我们的查询速度，所以要提高我们整个分页算法的速度，就要彻底改造 NOT IN，同其他方法来替代它。

我们知道，几乎任何字段，我们都可以通过 max(字段)或 min(字段)来提取某个字段中的最大或最小值，所以如果这个字段不重复，那么就可以利用这些不重复的字段的 max 或 min 作为分水岭，使其成为分页算法中分开每页的参照物。在这里，我们可以用操作符">"或"<"号来完成这个使命，使查询语句符合 SARG 形式。如:

```
Select top 10 * from table1 where id>200
```

于是就有了如下分页方案:

```
select top 页大小 *
from table1
where id>
(select max (id) from
(select top ((页码-1)*页大小) id from table1 order by id) as T
)
order by id
```

在选择即不重复值，又容易分辨大小的列时，我们通常会选择主键。下表列出了笔者用有着 1000 万数据的办公自动化系统中的表，在以 GID(GID 是主键，但并不是聚集索引。)为排序列、提取 gid,fariqi,title 字段，分别以第 1、10、100、500、1000、1 万、10 万、25 万、50 万页为例，测试以上三种分页方案的执行速度:(单位:毫秒)

页 码	方案 1	方案 2	方案 3
1	60	30	76
10	46	16	63
100	1076	720	130

500	540	12943	83
1000	17110	470	250
1 万	24796	4500	140
10 万	38326	42283	1553
25 万	28140	128720	2330
50 万	121686	127846	7168

从上表中，我们可以看出，三种存储过程在执行 100 页以下的分页命令时，都是可以信任的，速度都很好。但第一种方案在执行分页 1000 页以上后，速度就降了下来。第二种方案大约是在执行分页 1 万页以上后速度开始降了下来。而第三种方案却始终没有大的降势，后劲仍然很足。

在确定了第三种分页方案后，我们可以据此写一个存储过程。大家知道 SQL SERVER 的存储过程是事先编译好的 SQL 语句，它的执行效率要比通过 WEB 页面传来的 SQL 语句的执行效率要高。下面的存储过程不仅含有分页方案，还会根据页面传来的参数来确定是否进行数据总数统计。

```
-- 获取指定页的数据
CREATE PROCEDURE pagination3
@tblName    varchar(255),        -- 表名
@strGetFields varchar(1000) = '*',  -- 需要返回的列
@fldName varchar(255)='',       -- 排序的字段名
@PageSize    int = 10,          -- 页尺寸
@PageIndex   int = 1,           -- 页码
@doCount   bit = 0,   -- 返回记录总数, 非 0 值则返回
@OrderType bit = 0,  -- 设置排序类型, 非 0 值则降序
@strWhere   varchar(1500) = ''  -- 查询条件 (注意: 不要加 where)
AS
declare @strSQL    varchar(5000)       -- 主语句
declare @strTmp    varchar(110)        -- 临时变量
declare @strOrder varchar(400)        -- 排序类型


if @doCount != 0
  begin
    if @strWhere !=''
    set @strSQL = "select count(*) as Total from [" + @tblName + "] where "+@strWhere
    else
    set @strSQL = "select count(*) as Total from [" + @tblName + "]"
end
--以上代码的意思是如果@doCount 传递过来的不是 0，就执行总数统计。以下的所有代码都是
@doCount 为 0 的情况
else
begin
```

```
if @OrderType != 0
begin
    set @strTmp = "<(select min"
set @strOrder = " order by [" + @fldName +"] desc"
--如果@OrderType 不是 0，就执行降序，这句很重要！
end
else
begin
    set @strTmp = ">(select max"
    set @strOrder = " order by [" + @fldName +"] asc"
end


if @PageIndex = 1
begin
    if @strWhere != ''
    set @strSQL = "select top " + str(@PageSize) +" "+@strGetFields+ "   from [" +
@tblName + "] where " + @strWhere + " " + @strOrder
     else
     set @strSQL = "select top " + str(@PageSize) +" "+@strGetFields+ "   from ["+
@tblName + "] "+ @strOrder
--如果是第一页就执行以上代码，这样会加快执行速度
end
else
begin
--以下代码赋予了@strSQL 以真正执行的 SQL 代码
set @strSQL = "select top " + str(@PageSize) +" "+@strGetFields+ "   from ["
    + @tblName + "] where [" + @fldName + "]" + @strTmp + "(["+ @fldName + "])
from (select top " + str((@PageIndex-1)*@PageSize) + " ["+ @fldName + "] from [" +
@tblName + "]" + @strOrder + ") as tblTmp)"+ @strOrder

if @strWhere != ''
    set @strSQL = "select top " + str(@PageSize) +" "+@strGetFields+ "   from ["
        + @tblName + "] where [" + @fldName + "]" + @strTmp + "(["
        + @fldName + "]) from (select top " + str((@PageIndex-1)*@PageSize) + " ["
        + @fldName + "] from [" + @tblName + "] where " + @strWhere + " "
        + @strOrder + ") as tblTmp) and " + @strWhere + " " + @strOrder
end
end
exec (@strSQL)
GO
```

上面的这个存储过程是一个通用的存储过程，其注释已写在其中了。

在大数据量的情况下，特别是在查询最后几页的时候，查询时间一般不会超过 9 秒;而用其他存储过程，在实践中就会导致超时，所以这个存储过程非常适用于大容量数据库的查询。

笔者希望能够通过对以上存储过程的解析，能给大家带来一定的启示，并给工作带来一定的效率提升，同时希望同行提出更优秀的实时数据分页算法。

四、聚集索引的重要性和如何选择聚集索引

在上一节的标题中，笔者写的是:实现小数据量和海量数据的通用分页显示存储过程。这是因为在将本存储过程应用于"办公自动化"系统的实践中时，笔者发现这第三种存储过程在小数据量的情况下，有如下现象:

1、分页速度一般维持在 1 秒和 3 秒之间。

2、在查询最后一页时，速度一般为 5 秒至 8 秒，哪怕分页总数只有 3 页或 30 万页。

虽然在超大容量情况下，这个分页的实现过程是很快的，但在分前几页时，这个 1-3 秒的速度比起第一种甚至没有经过优化的分页方法速度还要慢，借用户的话说就是"还没有 ACCESS 数据库速度快"，这个认识足以导致用户放弃使用您开发的系统。

笔者就此分析了一下，原来产生这种现象的症结是如此的简单，但又如此的重要:排序的字段不是聚集索引!

本篇文章的题目是:"查询优化及分页算法方案"。笔者只所以把"查询优化"和"分页算法"这两个联系不是很大的论题放在一起，就是因为二者都需要一个非常重要的东西——聚集索引。

在前面的讨论中我们已经提到了，聚集索引有两个最大的优势:

1、以最快的速度缩小查询范围。

2、以最快的速度进行字段排序。

第 1 条多用在查询优化时，而第 2 条多用在进行分页时的数据排序。

而聚集索引在每个表内又只能建立一个，这使得聚集索引显得更加的重要。聚集索引的挑选可以说是实现"查询优化"和"高效分页"的最关键因素。

但要既使聚集索引列既符合查询列的需要，又符合排序列的需要，这通常是一个矛盾。

笔者前面"索引"的讨论中，将 fariqi，即用户发文日期作为了聚集索引的起始列，日期的精确度为"日"。这种作法的优点，前面已经提到了，在进行划时间段的快速查询中，比用 ID 主键列有很大的优势。

但在分页时，由于这个聚集索引列存在着重复记录，所以无法使用 max 或 min 来最为分页的参照物，进而无法实现更为高效的排序。而如果将 ID 主键列作为聚集索引，那么聚集索引除了用以排序之外，没有任何用处，实际上是浪费了聚集索引这个宝贵的资源。

为解决这个矛盾，笔者后来又添加了一个日期列，其默认值为 getdate()。用户在写入记录时，这个列自动写入当时的时间，时间精确到毫秒。即使这样，为了避免可能性很小的重合，还要在此列上创建 UNIQUE 约束。将此日期列作为聚集索引列。

有了这个时间型聚集索引列之后，用户就既可以用这个列查找用户在插入数据时的某个时间段的查询，又可以作为唯一列来实现 max 或 min，成为分页算法的参照物。

经过这样的优化，笔者发现，无论是大数据量的情况下还是小数据量的情况下，分页速度一般都是几十毫秒，甚至 0 毫秒。而用日期段缩小范围的查询速度比原来也没有任何迟钝。

聚集索引是如此的重要和珍贵，所以笔者总结了一下，一定要将聚集索引建立在:

1、您最频繁使用的、用以缩小查询范围的字段上;

2、您最频繁使用的、需要排序的字段上。

结束语:

本篇文章汇集了笔者近段在使用数据库方面的心得，是在做"办公自动化"系统时实践经验的积累。希望这篇文章不仅能够给大家的工作带来一定的帮助，也希望能让大家能够体会到分析问题的方法;最重要的是，希望这篇文章能够抛砖引玉，掀起大家的学习和讨论的兴趣，以共同促进，共同为公安科技强警事业和金盾工程做出自己最大的努力。

最后需要说明的是，在试验中，我发现用户在进行大数据量查询的时候，对数据库速度影响最大的不是内存大小，而是 CPU。在我的 P4 2.4 机器上试验的时候，查看"资源管理器"，CPU 经常出现持续到 100%的现象，而内存用量却并没有改变或者说没有大的改变。即使在我们的 HP ML 350 G3 服务器上试验时，CPU 峰值也能达到 90%，一般持续在 70%左右。

本文的试验数据都是来自我们的 HP ML 350 服务器。服务器配置:双 Inter Xeon 超线程 CPU 2.4G，内存 1G，操作系统 Windows Server 2003 Enterprise Edition，数据库 SQL Server 2000 SP3。