**[大数据量高并发的数据库优化](http://www.cnblogs.com/chuncn/archive/2009/04/21/1440233.html)**

一、数据库结构的设计

    如果不能设计一个合理的数据库模型，不仅会增加客户端和服务器段程序的编程和维护的难度，而且将会影响系统实际运行的性能。所以，在一个系统开始实施之前，完备的数据库模型的设计是必须的。

    在一个系统分析、设计阶段，因为数据量较小，负荷较低。我们往往只注意到功能的实现，而很难注意到性能的薄弱之处，等到系统投入实际运行一段时间后，才发现系统的性能在降低，这时再来考虑提高系统性能则要花费更多的人力物力，而整个系统也不可避免的形成了一个打补丁工程。

    所以在考虑整个系统的流程的时候，我们必须要考虑，在高并发大数据量的访问情况下，我们的系统会不会出现极端的情况。（例如：对外统计系统在7月16日出现的数据异常的情况，并发大数据量的的访问造成，数据库的响应时间不能跟上数据刷新的速度造成。具体情况是：在日期临界时（00：00：00），判断数据库中是否有当前日期的记录，没有则插入一条当前日期的记录。在低并发访问的情况下，不会发生问题，但是当日期临界时的访问量相当大的时候，在做这一判断的时候，会出现多次条件成立，则数据库里会被插入多条当前日期的记录，从而造成数据错误。），数据库的模型确定下来之后，我们有必要做一个系统内数据流向图，分析可能出现的瓶颈。

    为了保证数据库的一致性和完整性，在逻辑设计的时候往往会设计过多的表间关联，尽可能的降低数据的冗余。（例如用户表的地区，我们可以把地区另外存放到一个地区表中）如果数据冗余低，数据的完整性容易得到保证，提高了数据吞吐速度，保证了数据的完整性，清楚地表达数据元素之间的关系。而对于多表之间的关联查询（尤其是大数据表）时，其性能将会降低，同时也提高了客户端程序的编程难度，因此，物理设计需折衷考虑，根据业务规则，确定对关联表的数据量大小、数据项的访问频度，对此类数据表频繁的关联查询应适当提高数据冗余设计但增加了表间连接查询的操作，也使得程序的变得复杂，为了提高系统的响应时间，合理的数据冗余也是必要的。设计人员在设计阶段应根据系统操作的类型、频度加以均衡考虑。  
   另外，最好不要用自增属性字段作为主键与子表关联。不便于系统的迁移和数据恢复。对外统计系统映射关系丢失（\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*）。

    原来的表格必须可以通过由它分离出去的表格重新构建。使用这个规定的好处是，你可以确保不会在分离的表格中引入多余的列，所有你创建的表格结构都与它们的实际需要一样大。应用这条规定是一个好习惯，不过除非你要处理一个非常大型的数据，否则你将不需要用到它。（例如一个通行证系统，我可以将USERID，USERNAME，USERPASSWORD，单独出来作个表，再把USERID作为其他表的外键）

表的设计具体注意的问题：

    1、数据行的长度不要超过8020字节，如果超过这个长度的话在物理页中这条数据会占用两行从而造成存储碎片，降低查询效率。  
    2、能够用数字类型的字段尽量选择数字类型而不用字符串类型的（电话号码），这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接回逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

    3、对于不可变字符类型char和可变字符类型varchar 都是8000字节,char查询快，但是耗存储空间，varchar查询相对慢一些但是节省存储空间。在设计字段的时候可以灵活选择，例如用户名、密码等长度变化不大的字段可以选择CHAR，对于评论等长度变化大的字段可以选择VARCHAR。

    4、字段的长度在最大限度的满足可能的需要的前提下，应该尽可能的设得短一些，这样可以提高查询的效率，而且在建立索引的时候也可以减少资源的消耗。

二、查询的优化   
  
保证在实现功能的基础上，尽量减少对数据库的访问次数；通过搜索参数，尽量减少对表的访问行数,最小化结果集，从而减轻网络负担；能够分开的操作尽量分开处理，提高每次的响应速度；在数据窗口使用SQL时，尽量把使用的索引放在选择的首列；算法的结构尽量简单；在查询时，不要过多地使用通配符如SELECT \* FROM T1语句，要用到几列就选择几列如：SELECT COL1,COL2 FROM T1；在可能的情况下尽量限制尽量结果集行数如：SELECT TOP 300 COL1,COL2,COL3 FROM T1,因为某些情况下用户是不需要那么多的数据的。     
在没有建索引的情况下，数据库查找某一条数据，就必须进行全表扫描了，对所有数据进行一次遍历，查找出符合条件的记录。在数据量比较小的情况下，也许看不出明显的差别，但是当数据量大的情况下，这种情况就是极为糟糕的了。  
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，一个值的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。   
    所以，优化查询最重要的就是，尽量使语句符合查询优化器的规则避免全表扫描而使用索引查询。

具体要注意的：

1.应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：  
select id from t where num is null  
可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：  
select id from t where num=0

2.应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。优化器将无法通过索引来确定将要命中的行数,因此需要搜索该表的所有行。

3.应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：  
select id from t where num=10 or num=20  
可以这样查询：  
select id from t where num=10  
union all  
select id from t where num=20

4.in 和 not in 也要慎用，因为IN会使系统无法使用索引,而只能直接搜索表中的数据。如：  
select id from t where num in(1,2,3)  
对于连续的数值，能用 between 就不要用 in 了：  
select id from t where num between 1 and 3

5.尽量避免在索引过的字符数据中，使用非打头字母搜索。这也使得引擎无法利用索引。   
见如下例子：   
SELECT \* FROM T1 WHERE NAME LIKE ‘%L%’   
SELECT \* FROM T1 WHERE SUBSTING(NAME,2,1)=’L’   
SELECT \* FROM T1 WHERE NAME LIKE ‘L%’   
即使NAME字段建有索引，前两个查询依然无法利用索引完成加快操作，引擎不得不对全表所有数据逐条操作来完成任务。而第三个查询能够使用索引来加快操作。

6.必要时强制查询优化器使用某个索引，如在 where 子句中使用参数，也会导致全表扫描。因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：  
select id from t where [num=@num](mailto:num=@num)  
可以改为强制查询使用索引：  
select id from t with(index(索引名)) where [num=@num](mailto:num=@num)

7.应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：  
SELECT \* FROM T1 WHERE F1/2=100   
应改为:   
SELECT \* FROM T1 WHERE F1=100\*2

SELECT \* FROM RECORD WHERE SUBSTRING(CARD\_NO,1,4)=’5378’   
应改为:   
SELECT \* FROM RECORD WHERE CARD\_NO LIKE ‘5378%’

SELECT member\_number, first\_name, last\_name FROM members   
WHERE DATEDIFF(yy,datofbirth,GETDATE()) > 21   
应改为:   
SELECT member\_number, first\_name, last\_name FROM members   
WHERE dateofbirth < DATEADD(yy,-21,GETDATE())   
即：任何对列的操作都将导致表扫描，它包括数据库函数、计算表达式等等，查询时要尽可能将操作移至等号右边。

8.应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：  
select id from t where substring(name,1,3)='abc'--name以abc开头的id  
select id from t where datediff(day,createdate,'2005-11-30')=0--‘2005-11-30’生成的id  
应改为:  
select id from t where name like 'abc%'  
select id from t where createdate>='2005-11-30' and createdate<'2005-12-1'

9.不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

10.在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

11.很多时候用 exists是一个好的选择：  
elect num from a where num in(select num from b)  
用下面的语句替换：  
select num from a where exists(select 1 from b where num=a.num)

SELECT SUM(T1.C1)FROM T1 WHERE(   
(SELECT COUNT(\*)FROM T2 WHERE T2.C2=T1.C2>0)   
SELECT SUM(T1.C1) FROM T1WHERE EXISTS(   
SELECT \* FROM T2 WHERE T2.C2=T1.C2)   
两者产生相同的结果，但是后者的效率显然要高于前者。因为后者不会产生大量锁定的表扫描或是索引扫描。

如果你想校验表里是否存在某条纪录，不要用count(\*)那样效率很低，而且浪费服务器资源。可以用EXISTS代替。如：   
IF (SELECT COUNT(\*) FROM table\_name WHERE column\_name = 'xxx')   
可以写成：   
IF EXISTS (SELECT \* FROM table\_name WHERE column\_name = 'xxx')

经常需要写一个T\_SQL语句比较一个父结果集和子结果集，从而找到是否存在在父结果集中有而在子结果集中没有的记录，如：   
SELECT a.hdr\_key FROM hdr\_tbl a---- tbl a 表示tbl用别名a代替   
WHERE NOT EXISTS (SELECT \* FROM dtl\_tbl b WHERE a.hdr\_key = b.hdr\_key)   
SELECT a.hdr\_key FROM hdr\_tbl a   
LEFT JOIN dtl\_tbl b ON a.hdr\_key = b.hdr\_key WHERE b.hdr\_key IS NULL   
SELECT hdr\_key FROM hdr\_tbl   
WHERE hdr\_key NOT IN (SELECT hdr\_key FROM dtl\_tbl)   
三种写法都可以得到同样正确的结果，但是效率依次降低。

12.尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。

13.避免频繁创建和删除临时表，以减少系统表资源的消耗。

14.临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。

15.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。

16.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。   
  
17.在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF 。无需在执行存储过程和触发器的每个语句后向客户端发送 DONE\_IN\_PROC 消息。

18.尽量避免大事务操作，提高系统并发能力。

19.尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。   
  
20. 避免使用不兼容的数据类型。例如float和int、char和varchar、binary和varbinary是不兼容的。数据类型的不兼容可能使优化器无法执行一些本来可以进行的优化操作。例如:   
SELECT name FROM employee WHERE salary > 60000   
在这条语句中,如salary字段是money型的,则优化器很难对其进行优化,因为60000是个整型数。我们应当在编程时将整型转化成为钱币型,而不要等到运行时转化。

21.充分利用连接条件，在某种情况下，两个表之间可能不只一个的连接条件，这时在 WHERE 子句中将连接条件完整的写上，有可能大大提高查询速度。   
例：   
SELECT SUM(A.AMOUNT) FROM ACCOUNT A,CARD B WHERE A.CARD\_NO = B.CARD\_NO   
SELECT SUM(A.AMOUNT) FROM ACCOUNT A,CARD B WHERE A.CARD\_NO = B.CARD\_NO AND A.ACCOUNT\_NO=B.ACCOUNT\_NO   
第二句将比第一句执行快得多。

22、使用视图加速查询   
把表的一个子集进行排序并创建视图，有时能加速查询。它有助于避免多重排序 操作，而且在其他方面还能简化优化器的工作。例如：  
SELECT cust.name，rcvbles.balance，……other columns   
FROM cust，rcvbles   
WHERE cust.customer\_id = rcvlbes.customer\_id   
AND rcvblls.balance>0   
AND cust.postcode>“98000”   
ORDER BY cust.name

如果这个查询要被执行多次而不止一次，可以把所有未付款的客户找出来放在一个视图中，并按客户的名字进行排序：   
CREATE VIEW DBO.V\_CUST\_RCVLBES   
AS   
SELECT cust.name，rcvbles.balance，……other columns   
FROM cust，rcvbles   
WHERE cust.customer\_id = rcvlbes.customer\_id   
AND rcvblls.balance>0   
ORDER BY cust.name   
然后以下面的方式在视图中查询：   
SELECT ＊ FROM V\_CUST\_RCVLBES   
WHERE postcode>“98000”   
视图中的行要比主表中的行少，而且物理顺序就是所要求的顺序，减少了磁盘I/O，所以查询工作量可以得到大幅减少。

23、能用DISTINCT的就不用GROUP BY   
SELECT OrderID FROM Details WHERE UnitPrice > 10 GROUP BY OrderID   
可改为：   
SELECT DISTINCT OrderID FROM Details WHERE UnitPrice > 10

24.能用UNION ALL就不要用UNION   
UNION ALL不执行SELECT DISTINCT函数，这样就会减少很多不必要的资源   
  
35.尽量不要用SELECT INTO语句。   
SELECT INOT 语句会导致表锁定，阻止其他用户访问该表。

    上面我们提到的是一些基本的提高查询速度的注意事项,但是在更多的情况下,往往需要反复试验比较不同的语句以得到最佳方案。最好的方法当然是测试，看实现相同功能的SQL语句哪个执行时间最少，但是数据库中如果数据量很少，是比较不出来的，这时可以用查看执行计划，即：把实现相同功能的多条SQL语句考到查询分析器，按CTRL+L看查所利用的索引，表扫描次数（这两个对性能影响最大），总体上看询成本百分比即可。   
  
三、算法的优化

尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。.使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。与临时表一样，游标并不是不可使用。对小型数据集使用 FAST\_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。  
　　游标提供了对特定集合中逐行扫描的手段，一般使用游标逐行遍历数据，根据取出的数据不同条件进行不同的操作。尤其对多表和大表定义的游标（大的数据集合）循环很容易使程序进入一个漫长的等特甚至死机。   
　 在有些场合，有时也非得使用游标，此时也可考虑将符合条件的数据行转入临时表中，再对临时表定义游标进行操作，可时性能得到明显提高。  
（例如：对内统计第一版）  
封装存储过程

四、建立高效的索引

　 创建索引一般有以下两个目的：维护被索引列的唯一性和提供快速访问表中数据的策略。大型数据库有两种索引即簇索引和非簇索引，一个没有簇索引的表是按堆结构存储数据，所有的数据均添加在表的尾部，而建立了簇索引的表，其数据在物理上会按照簇索引键的顺序存储，一个表只允许有一个簇索引，因此，根据B树结构，可以理解添加任何一种索引均能提高按索引列查询的速度，但会降低插入、更新、删除操作的性能，尤其是当填充因子（Fill Factor）较大时。所以对索引较多的表进行频繁的插入、更新、删除操作，建表和索引时因设置较小的填充因子，以便在各数据页中留下较多的自由空间，减少页分割及重新组织的工作。   
索引是从数据库中获取数据的最高效方式之一。95% 的数据库性能问题都可以采用索引技术得到解决。作为一条规则，我通常对逻辑主键使用唯一的成组索引，对系统键（作为存储过程）采用唯一的非成组索引，对任何外键列[字段]采用非成组索引。不过，索引就象是盐，太多了菜就咸了。你得考虑数据库的空间有多大，表如何进行访问，还有这些访问是否主要用作读写。   
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”。   
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。

（一）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。

（二）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。

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）   
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。

（三）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。

分类: [数据库](http://www.cnblogs.com/chuncn/category/101869.html)