sql语句优化

性能不理想的系统中除了一部分是因为应用程序的负载确实超过了服务器的实际处理能力外,更多的是因为系统存在大量的SQL语句需要优化。

为了获得稳定的执行性能，SQL语句越简单越好。对复杂的SQL语句，要设法对之进行简化。

常见的简化规则如下：  
   
1）不要有超过5个以上的表连接（JOIN）  
2）考虑使用临时表或表变量存放中间结果。  
3）少用子查询  
4）视图嵌套不要过深,一般视图嵌套不要超过2个为宜。

连接的表越多，其编译的时间和连接的开销也越大，性能越不好控制。

最好是把连接拆开成较小的几个部分逐个顺序执行。

优先执行那些能够大量减少结果的连接。

拆分的好处不仅仅是减少SQL Server优化的时间，更使得SQL语句能够以你可以预测的方式和顺序执行。

如果一定需要连接很多表才能得到数据，那么很可能意味着设计上的缺陷。

连接是outer join，非常不好。因为outer join意味着必须对左表或右表查询所有行。

如果表很大而没有相应的where语句，那么outer join很容易导致table scan或index scan。

要尽量使用inner join避免scan整个表。

优化建议：  
   
1）使用临时表存放t1表的结果,能大大减少logical reads（或返回行数）的操作要优先执行。

 仔细分析语句，你会发现where中的条件全是针对表t1的，所以直接使用上面的where子句查询表t1，然后把结果存放再临时表＃t1中：  
   
Select t1….. into #tt1 from t1 where…(和上面的where一样)  
   
2）再把＃tt1和其他表进行连接:  
   
Select #t1…  
Left outer join …  
Left outer join…  
   
   
3）修改 like 程序，去掉前置百分号。like语句却因为前置百分号而无法使用索引  
4）从系统设计的角度修改语句，去掉outer join。  
5）考虑组合索引或覆盖索引消除clustered index scan。  
   
上面1和2点建议立即消除了worktable，性能提高了几倍以上，效果非常明显。

1）限制结果集  
   
要尽量减少返回的结果行，包括行数和字段列数。

返回的结果越大，意味着相应的SQL语句的logical reads 就越大，对服务器的性能影响就越甚。

一个很不好的设计就是返回表的所有数据：  
   
Select \* from tablename  
   
即使表很小也会导致并发问题。更坏的情况是，如果表有上百万行的话，那后果将是灾难性的。

它不但可能带来极重的磁盘IO，更有可能把**[数据库](http://lib.csdn.net/base/mysql" \t "_blank" \o "MySQL知识库)**缓冲区中的其他缓存数据挤出，使得这些数据下次必须再从磁盘读取。

必须设计良好的SQL语句，使得其有where语句或TOP语句来限制结果集大小。

2）合理的表设计  
   
SQL Server 2005将支持表分区技术。利用表分区技术可以实现数据表的流动窗口功能。

在流动窗口中可以轻易的把历史数据移出，把新的数据加入，从而使表的大小基本保持稳定。  
   
另外，表的设计未必需要非常范式化。有一定的字段冗余可以增加SQL语句的效率，减少JOIN的数目，提高语句的执行速度。

3）OLAP和OLTP模块要分开  
   
OLAP和OLTP类型的语句是截然不同的。前者往往需要扫描整个表做统计分析，索引对这样的语句几乎没有多少用处。

索引只能够加快那些如sum，group by之类的聚合运算。因为这个原因，几乎很难对OLAP类型的SQL语句进行优化。

而OLTP语句则只需要访问表的很小一部分数据，而且这些数据往往可以从内存缓存中得到。

为了避免OLAP 和OLTP语句相互影响，这两类模块需要分开运行在不同服务器上。

因为OLAP语句几乎都是读取数据，没有更新和写入操作，所以一个好的经验是配置一台standby 服务器，然后OLAP只访问standby服务器。

4）使用存储过程  
   
可以考虑使用存储过程封装那些复杂的SQL语句或商业逻辑，这样做有几个好处。

一是存储过程的执行计划可以被缓存在内存中较长时间，减少了重新编译的时间。

二是存储过程减少了客户端和服务器的繁复交互。

三是如果程序发布后需要做某些改变你可以直接修改存储过程而不用修改程序，避免需要重新安装部署程序。

 索引优化

很多数据库系统性能不理想是因为系统没有经过整体优化，存在大量性能低下的SQL 语句。

这类SQL语句性能不好的首要原因是缺乏高效的索引。

没有索引除了导致语句本身运行速度慢外，更是导致大量的磁盘读写操作，使得整个系统性能都受之影响而变差。

解决这类系统的首要办法是优化这些没有索引或索引不够好的SQL语句。

创建索引的关键

优化SQL语句的关键是尽可能减少语句的logical reads。

这里说的logical reads是指语句执行时需要访问的单位为8K的数据页总数。

logical reads 越少，其需要的内存和CPU时间也就越少，语句执行速度就越快。

不言而喻，索引的最大好处是它可以极大减少SQL语句的logical reads数目，从而极大减少语句的执行时间。

创建索引的关键是索引要能够大大减少语句的logical reads。一个索引好不好，主要看它减少的logical reads多不多。

运行set statistics io命令可以得到SQL语句的logical reads信息。

set statistics io on

select au\_id,au\_lname ,au\_fname

from pubs..authors where au\_lname ='Green'

set statistics io on

如果Logical reads很大，而返回的行数很少，也即两者相差较大，那么往往意味者语句需要优化。

Logical reads中包含该语句从内存数据缓冲区中访问的页数和从物理磁盘读取的页数。

而physical reads表示那些没有驻留在内存缓冲区中需要从磁盘读取的数据页。

Read-ahead reads是SQL Server为了提高性能而产生的预读。预读可能会多读取一些数据。

优化的时候我们主要关注Logical Reads就可以了。

注意如果physical Reads或Read-ahead reads很大，那么往往意味着语句的执行时间（duration）里面会有一部分耗费在等待物理磁盘IO上。

二、单字段索引，组合索引和覆盖索引

单字段索引是指只有一个字段的索引，而组合索引指有多个字段构成的索引。

1． 对出现在where子句中的字段加索引

set statistics profile on

set statistics io on

go

select .... from tb where ...

go

set statistics profile off

set statistics io off

set statistics profile命令将输出语句的执行计划。

也许你会问，为什么不用SET SHOWPLAN\_ALL呢？使用SET SHOWPLAN\_ALL也是可以的。

不过set statistics profile输出的是SQL 语句的运行时候真正使用的执行计划，

而SET SHOWPLAN\_ALL输出的是预计（Estimate）的执行计划。

使用SET SHOWPLAN\_ALL是后面的语句并不会真正运行。

用了Table Scan，也就是对整个表进行了全表扫描。全表扫描的性能通常是很差的，要尽量避免。

如果上面的select语句是数据库系统经常运行的关键语句， 那么应该对它创建相应的索引。

创建索引的技巧之一是对经常出现在where条件中的字段创建索引

Table Scan也变成了Index Seek，性能极大提高

设法避免Table scan或Index scan是优化SQL 语句使用的常用技巧。通常Index Seek需要的logical reads比前两者要少得多。

2．组合索引

如果where语句中有多个字段，那么可以考虑创建组合索引。

组合索引中字段的顺序是非常重要的，越是唯一的字段越是要靠前。

另外，无论是组合索引还是单个列的索引，尽量不要选择那些唯一性很低的字段。

比如说，在只有两个值0和1的字段上建立索引没有多大意义。

所以如果对单字段进行索引，建议使用set statistics profile来验证索引确实被充分使用。logical reads越少的索引越好。

3．覆盖索引

覆盖索引能够使得语句不需要访问表仅仅访问索引就能够得到所有需要的数据。

因为聚集索引叶子节点就是数据所以无所谓覆盖与否，所以覆盖索引主要是针对非聚集索引而言。

执行计划中除了index seek外，还有一个Bookmark Lookup关键字。

Bookmark Lookup表示语句在访问索引后还需要对表进行额外的Bookmark Lookup操作才能得到数据。

也就是说为得到一行数据起码有两次IO，一次访问索引，一次访问基本表。

如果语句返回的行数很多，那么Bookmark Lookup操作的开销是很大的。

覆盖索引能够避免昂贵的Bookmark Lookup操作，减少IO的次数，提高语句的性能。

覆盖索引需要包含select子句和WHERE子句中出现的所有字段。Where语句中的字段在前面，select中的在后面。

logical reads，是大大减少了。Bookmark Lookup操作也消失了。所以创建覆盖索引是减少logical reads提升语句性能的非常有用的优化技巧。

实际上索引的创建原则是比较复杂的。有时候你无法在索引中包含了Where子句中所有的字段。

在考虑索引是否应该包含一个字段时，应考虑该字段在语句中的作用。

比如说如果经常以某个字段作为where条件作精确匹配返回很少的行，那么就绝对值得为这个字段建立索引。

再比如说，对那些非常唯一的字段如主键和外键，经常出现在group by，order by中的字段等等都值得创建索引。

问题1，是否值得在identity字段上建立聚集索引。

答案取决于identity 字段如何在语句中使用。如果你经常根据该字段搜索返回很少的行，那么在其上建立索引是值得的。

反之如果identity字段根本很少在语句中使用，那么就不应该对其建立任何索引。

问题2，一个表应该建立多少索引合适。

如果表的80％以上的语句都是读操作，那么索引可以多些。但是不要太多。

特别是不要对那些更新频繁的表其建立很多的索引。很少表有超过5个以上的索引。

过多的索引不但增加其占用的磁盘空间，也增加了SQL Server 维护索引的开销。

问题4：为什么SQL Server 在执行计划中没有使用你认为应该使用的索引？原因是多样的。

一种原因是该语句返回的结果超过了表的20％数据，使得SQL Server 认为scan比seek更有效。

另一种原因可能是表字段的statistics过期了，不能准确反映数据的分布情况。

你可以使用命令UPDATE STATISTICS tablename with FULLSCAN来更新它。

只有同步的准确的statistics才能保证SQL Server 产生正确的执行计划。

过时的老的statistics常会导致SQL Server生成不够优化的甚至愚蠢的执行计划。

所以如果你的表频繁更新，而你又觉得和之相关的SQL语句运行缓慢，不妨试试UPDATE STATISTIC with FULLSCAN 语句。

问题5、什么使用聚集索引，什么时候使用非聚集索引

在SQL Server 中索引有聚集索引和非聚集索引两种。它们的主要差别是前者的索引叶子就是数据本身，而后者的叶子节点包含的是指向数据的书签（即数据行号或聚集索引的key）。

对一个表而言聚集索引只能有一个，而非聚集索引可以有多个。

只是聚集索引没有Bookmark Lookup操作。

什么时候应该使用聚集索引?  什么时候使用非聚集索引? 取决于应用程序的访问模式。

我的建议是在那些关键的字段上使用聚集索引。一个表一般都需要建立一个聚集索引。

对于什么时候使用聚集索引，SQL Server 2000联机手册中有如下描述：

在创建聚集索引之前，应先了解您的数据是如何被访问的。可考虑将聚集索引用于：

包含大量非重复值的列。

使用下列运算符返回一个范围值的查询：BETWEEN、>、>=、< 和 <=。

被连续访问的列。

返回大型结果集的查询。

经常被使用联接或 GROUP BY 子句的查询访问的列；一般来说，这些是外键列。

对 ORDER BY 或 GROUP BY 子句中指定的列进行索引，可以使 SQL Server 不必对数据进行排序，因为这些行已经排序。这样可以提高查询性能。

OLTP 类型的应用程序，这些程序要求进行非常快速的单行查找（一般通过主键）。应在主键上创建聚集索引。

聚集索引不适用于：

频繁更改的列

这将导致整行移动（因为 SQL Server 必须按物理顺序保留行中的数据值）。这一点要特别注意，因为在大数据量事务处理系统中数据是易失的。

宽键

来自聚集索引的键值由所有非聚集索引作为查找键使用，因此存储在每个非聚集索引的叶条目内。

总结：

如何使一个性能缓慢的系统运行更快更高效，不但需要整体分析数据库系统，找出系统的性能瓶颈，更需要优化数据库系统发出的SQL 语句。

一旦找出关键的SQL 语句并加与优化，性能问题就会迎刃而解。



《 数据库技术内幕 》

处理百万级以上的数据提高查询速度的方法：  
  
 **1**.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。  
  
 **2**.对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。  
  
 **3**.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**  
 **4**.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**  
 **5**.下面的查询也将导致全表扫描：(不能前置百分号)  
     select id from t where name like ‘%abc%’  
    若要提高效率，可以考虑全文检索。  
  
 **6**.in 和 not in 也要慎用，否则会导致全表扫描，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**

select xx,phone FROM send  a JOIN (  
 select '13891030091' phone  union select '13992085916' …………  UNION  SELECT '13619100234' ) b   
  on  a.Phone=b.phone  
--替代下面  很多数据隔开的时候  
in('13891030091','13992085916','13619100234'…………)

**7**.如果在 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     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)  
  
 **8**.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 id from t where num/**2**=**100**  
     应改为:  
     select id from t where num=**100**\***2**  
 **9**.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**′  
  
 **10**.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。  
  
 **11**.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。  
  
 **12**.不要写一些没有意义的查询，如需要生成一个空表结构：  
     select col1,col2 into #t from t where **1**=**0**  
     这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：  
     create table #t(…)  
  
 **13**.很多时候用 exists 代替 in 是一个好的选择：  
     select 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)  
  
 **14**.并不是所有索引对查询都有效，SQL是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，如一表中有字段 sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。  
  
 **15**.索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有 必要。  
  
 **16**.应尽可能的避免更新 clustered 索引数据列，因为 clustered 索引数据列的顺序就是表记录的物理存储顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。  
  
 **17**.尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，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时会 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。  
  
 **18**.尽可能的使用 varchar/nvarchar 代替 char/nchar ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。  
  
 **19**.任何地方都不要使用 select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段。  
  
 **20**.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）。  
  
 **21**.避免频繁创建和删除临时表，以减少系统表资源的消耗。  
  
 **22**.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。  
  
 **23**.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。  
  
 **24**.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。  
  
 **25**.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。  
  
 **26**.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。  
  
 **27**.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。  
  
 **28**.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。  
  
 **29**.尽量避免向客户端返回**[大数据](http://lib.csdn.net/base/hadoop" \t "_blank" \o "Hadoop知识库)**量，若数据量过大，应该考虑相应需求是否合理。  
  
 **30**.尽量避免大事务操作，提高系统并发能力。  
  
  
  
查询速度慢的原因：  
  
**1**、没有索引或者没有用到索引(这是查询慢最常见的问题，是程序设计的缺陷)   
   
**2**、I/O吞吐量小，形成了瓶颈效应。    
  
**3**、没有创建计算列导致查询不优化。   
   
**4**、内存不足    
  
**5**、网络速度慢    
  
**6**、查询出的数据量过大（可以采用多次查询，其他的方法降低数据量）    
  
**7**、锁或者死锁(这也是查询慢最常见的问题，是程序设计的缺陷)    
  
**8**、sp\_lock,sp\_who,活动的用户查看,原因是读写竞争资源。  
    
**9**、返回了不必要的行和列    
  
**10**、查询语句不好，没有优化    
  
可以通过如下方法来优化查询    
  
**1**、把数据、日志、索引放到不同的I/O设备上，增加读取速度，以前可以将Tempdb应放在RAID0上，SQL2000不在支持。数据量（尺寸）越大，提高I/O越重要.    
  
**2**、纵向、横向分割表，减少表的尺寸(sp\_spaceuse)    
  
**3**、升级硬件    
  
**4**、根据查询条件,建立索引,优化索引、优化访问方式，限制结果集的数据量。注意填充因子要适当（最好是使用默认值0）。索引应该尽量小，使用字节数小的列建索引好（参照索引的创建）,不要对有限的几个值的字段建单一索引如性别字段    
  
**5**、提高网速;    
  
**6**、扩大服务器的内存,Windows   2000和SQL   server   2000能支持4-8G的内存。配置虚拟内存：虚拟内存大小应基于计算机上并发运行的服务进行配置。运行   Microsoft   SQL   Server?   **2000**   时，可考虑将虚拟内存大小设置为计算机中安装的物理内存的   **1.5**   倍。如果另外安装了全文检索功能，并打算运行   Microsoft   搜索服务以便执行全文索引和查询，可考虑：将虚拟内存大小配置为至少是计算机中安装的物理内存的   **3**   倍。将   SQL   Server   max   server   memory   服务器配置选项配置为物理内存的   **1.5**   倍（虚拟内存大小设置的一半）。    
  
**7**、增加服务器CPU个数;但是必须明白并行处理串行处理更需要资源例如内存。使用并行还是串行程是MsSQL自动评估选择的。单个任务分解成多个任务，就可以在处理器上运行。例如耽搁查询的排序、连接、扫描和GROUP   BY字句同时执行，SQL   SERVER根据系统的负载情况决定最优的并行等级，复杂的需要消耗大量的CPU的查询最适合并行处理。但是更新操作UPDATE,INSERT， DELETE还不能并行处理。    
  
**8**、如果是使用like进行查询的话，简单的使用index是不行的，但是全文索引，耗空间。   like   'a%'   使用索引   like   '%a'   不使用索引用   like   '%a%'   查询时，查询耗时和字段值总长度成正比,所以不能用CHAR类型，而是VARCHAR。对于字段的值很长的建全文索引。    
  
**9**、DB   Server   和APPLication   Server   分离；OLTP和OLAP分离    
  
**10**、分布式分区视图可用于实现数据库服务器联合体。联合体是一组分开管理的服务器，但它们相互协作分担系统的处理负荷。这种通过分区数据形成数据库服务器联合体的机制能够扩大一组服务器，以支持大型的多层   Web   站点的处理需要。有关更多信息，参见设计联合数据库服务器。（参照SQL帮助文件'分区视图'）    
    a、在实现分区视图之前，必须先水平分区表    
    b、在创建成员表后，在每个成员服务器上定义一个分布式分区视图，并且每个视图具有相同的名称。这样，引用分布式分区视图名的查询可以在任何一个成员服务器上运行。系统操作如同每个成员服务器上都有一个原始表的复本一样，但其实每个服务器上只有一个成员表和一个分布式分区视图。数据的位置对应用程序是透明的。    
  
**11**、重建索引   DBCC   REINDEX   ,DBCC   INDEXDEFRAG,收缩数据和日志   DBCC   SHRINKDB,DBCC   SHRINKFILE.   设置自动收缩日志.对于大的数据库不要设置数据库自动增长，它会降低服务器的性能。   在T-sql的写法上有很大的讲究，下面列出常见的要点：首先，DBMS处理查询计划的过程是这样的：    
    **1**、   查询语句的词法、语法检查            
    **2**、   将语句提交给DBMS的查询优化器    
    **3**、   优化器做代数优化和存取路径的优化    
    **4**、   由预编译模块生成查询规划    
    **5**、   然后在合适的时间提交给系统处理执行    
    **6**、   最后将执行结果返回给用户其次，看一下SQL   SERVER的数据存放的结构：一个页面的大小为8K(**8060**)字节，8个页面为一个盘区，按照B树存放。    
  
**12**、Commit和rollback的区别   Rollback:回滚所有的事物。   Commit:提交当前的事物.   没有必要在动态SQL里写事物，如果要写请写在外面如：   begin   tran   exec(@s)   commit   trans   或者将动态SQL   写成函数或者存储过程。    
  
**13**、在查询Select语句中用Where字句限制返回的行数,避免表扫描,如果返回不必要的数据，浪费了服务器的I/O资源，加重了网络的负担降低性能。如果表很大，在表扫描的期间将表锁住，禁止其他的联接访问表,后果严重。    
  
**14**、SQL的注释申明对执行没有任何影响    
  
**15**、尽可能不使用游标，它占用大量的资源。如果需要row-by-row地执行，尽量采用非光标技术,如：在客户端循环，用临时表，Table变量，用子查询，用Case语句等等。游标可以按照它所支持的提取选项进行分类：   只进   必须按照从第一行到最后一行的顺序提取行。FETCH   NEXT   是唯一允许的提取操作,也是默认方式。可滚动性   可以在游标中任何地方随机提取任意行。游标的技术在SQL2000下变得功能很强大，他的目的是支持循环。    
有四个并发选项    
READ\_ONLY：不允许通过游标定位更新(Update)，且在组成结果集的行中没有锁。    
OPTIMISTIC   WITH   valueS:乐观并发控制是事务控制理论的一个标准部分。乐观并发控制用于这样的情形，即在打开游标及更新行的间隔中，只有很小的机会让第二个用户更新某一行。当某个游标以此选项打开时，没有锁控制其中的行，这将有助于最大化其处理能力。如果用户试图修改某一行，则此行的当前值会与最后一次提取此行时获取的值进行比较。如果任何值发生改变，则服务器就会知道其他人已更新了此行，并会返回一个错误。如果值是一样的，服务器就执行修改。   选择这个并发选项OPTIMISTIC   WITH   ROW   VERSIONING:此乐观并发控制选项基于行**[版本控制](http://lib.csdn.net/base/git" \t "_blank" \o "Git知识库)**。使用行版本控制，其中的表必须具有某种版本标识符，服务器可用它来确定该行在读入游标后是否有所更改。    
在   SQL   Server   中，这个性能由   timestamp   数据类型提供，它是一个二进制数字，表示数据库中更改的相对顺序。每个数据库都有一个全局当前时间戳值：**@@DBTS**。每次以任何方式更改带有   timestamp   列的行时，SQL   Server   先在时间戳列中存储当前的   **@@DBTS**   值，然后增加   **@@DBTS**   的值。如果某   个表具有   timestamp   列，则时间戳会被记到行级。服务器就可以比较某行的当前时间戳值和上次提取时所存储的时间戳值，从而确定该行是否已更新。服务器不必比较所有列的值，只需比较   timestamp   列即可。如果应用程序对没有   timestamp   列的表要求基于行版本控制的乐观并发，则游标默认为基于数值的乐观并发控制。    
SCROLL   LOCKS   这个选项实现悲观并发控制。在悲观并发控制中，在把数据库的行读入游标结果集时，应用程序将试图锁定数据库行。在使用服务器游标时，将行读入游标时会在其上放置一个更新锁。如果在事务内打开游标，则该事务更新锁将一直保持到事务被提交或回滚；当提取下一行时，将除去游标锁。如果在事务外打开游标，则提取下一行时，锁就被丢弃。因此，每当用户需要完全的悲观并发控制时，游标都应在事务内打开。更新锁将阻止任何其它任务获取更新锁或排它锁，从而阻止其它任务更新该行。    
然而，更新锁并不阻止共享锁，所以它不会阻止其它任务读取行，除非第二个任务也在要求带更新锁的读取。滚动锁根据在游标定义的   SELECT   语句中指定的锁提示，这些游标并发选项可以生成滚动锁。滚动锁在提取时在每行上获取，并保持到下次提取或者游标关闭，以先发生者为准。下次提取时，服务器为新提取中的行获取滚动锁，并释放上次提取中行的滚动锁。滚动锁独立于事务锁，并可以保持到一个提交或回滚操作之后。如果提交时关闭游标的选项为关，则   COMMIT   语句并不关闭任何打开的游标，而且滚动锁被保留到提交之后，以维护对所提取数据的隔离。所获取滚动锁的类型取决于游标并发选项和游标   SELECT   语句中的锁提示。    
锁提示   只读   乐观数值   乐观行版本控制   锁定无提示   未锁定   未锁定   未锁定   更新   NOLOCK   未锁定   未锁定   未锁定   未锁定   HOLDLOCK   共享   共享   共享   更新   UPDLOCK   错误   更新   更新   更新   TABLOCKX   错误   未锁定   未锁定   更新其它   未锁定   未锁定   未锁定   更新   \*指定   NOLOCK   提示将使指定了该提示的表在游标内是只读的。    
  
**16**、用Profiler来跟踪查询，得到查询所需的时间，找出SQL的问题所在;用索引优化器优化索引    
  
**17**、注意UNion和UNion   all   的区别。UNION   all好    
  
**18**、注意使用DISTINCT，在没有必要时不要用，它同UNION一样会使查询变慢。重复的记录在查询里是没有问题的    
  
**19**、查询时不要返回不需要的行、列    
  
**20**、用sp\_configure   'query   governor   cost   limit'或者SET   QUERY\_GOVERNOR\_COST\_LIMIT来限制查询消耗的资源。当评估查询消耗的资源超出限制时，服务器自动取消查询,在查询之前就扼杀掉。 SET   LOCKTIME设置锁的时间    
  
**21**、用select   top   **100**   /   **10**   Percent   来限制用户返回的行数或者SET   ROWCOUNT来限制操作的行    
  
**22**、在SQL2000以前，一般不要用如下的字句 “IS   NULL",   " <> ",   "!=",   "!> ",   "! <",   "NOT",   "NOT   EXISTS",   "NOT   IN",   "NOT   LIKE",   and   "LIKE   '%500'"，因为他们不走索引全是表扫描。  
也不要在WHere字句中的列名加函数，如Convert，substring等,如果必须用函数的时候，创建计算列再创建索引来替代.还可以变通写法：WHERE   SUBSTRING(firstname,**1**,**1**)   =   'm'改为WHERE   firstname   like   'm%'（索引扫描），一定要将函数和列名分开。并且索引不能建得太多和太大。  
NOT   IN会多次扫描表，使用EXISTS、NOT   EXISTS   ，IN   ,   LEFT   OUTER   JOIN   来替代，特别是左连接,而Exists比IN更快，最慢的是NOT操作.如果列的值含有空，以前它的索引不起作用，现在2000的优化器能够处理了。相同的是IS   NULL，“NOT",   "NOT   EXISTS",   "NOT   IN"能优化她，而” <> ”等还是不能优化，用不到索引。    
  
**23**、使用Query   Analyzer，查看SQL语句的查询计划和评估分析是否是优化的SQL。一般的20%的代码占据了80%的资源，我们优化的重点是这些慢的地方。    
  
**24**、如果使用了IN或者OR等时发现查询没有走索引，使用显示申明指定索引：   SELECT   \*   FROM   PersonMember   (INDEX   =   IX\_Title)   WHERE   processid   IN   (‘男’，‘女’)    
  
**25**、将需要查询的结果预先计算好放在表中，查询的时候再SELECT。这在SQL7.0以前是最重要的手段。例如医院的住院费计算。    
  
**26**、MIN()   和   MAX()能使用到合适的索引    
  
**27**、数据库有一个原则是代码离数据越近越好，所以优先选择Default,依次为Rules,Triggers,   Constraint（约束如外健主健CheckUNIQUE……,数据类型的最大长度等等都是约束）,Procedure.这样不仅维护工作小，编写程序质量高，并且执行的速度快。    
  
**28**、如果要插入大的二进制值到Image列，使用存储过程，千万不要用内嵌INsert来插入(不知**[Java](http://lib.csdn.net/base/javase" \t "_blank" \o "Java SE知识库)**是否)。因为这样应用程序首先将二进制值转换成字符串（尺寸是它的两倍），服务器受到字符后又将他转换成二进制值.存储过程就没有这些动作:   方法：Create   procedure   p\_insert   as   insert   into   table(Fimage)   values   (@image),   在前台调用这个存储过程传入二进制参数，这样处理速度明显改善。    
  
**29**、Between在某些时候比IN速度更快,Between能够更快地根据索引找到范围。用查询优化器可见到差别。   select   \*   from   chineseresume   where   title   in   ('男','女')   Select   \*   from   chineseresume   where   between   '男'   and   '女'   是一样的。由于in会在比较多次，所以有时会慢些。    
  
**30**、在必要是对全局或者局部临时表创建索引，有时能够提高速度，但不是一定会这样，因为索引也耗费大量的资源。他的创建同是实际表一样。    
  
**31**、不要建没有作用的事物例如产生报表时，浪费资源。只有在必要使用事物时使用它。    
  
**32**、用OR的字句可以分解成多个查询，并且通过UNION   连接多个查询。他们的速度只同是否使用索引有关,如果查询需要用到联合索引，用UNION   all执行的效率更高.多个OR的字句没有用到索引，改写成UNION的形式再试图与索引匹配。一个关键的问题是否用到索引。    
  
**33**、尽量少用视图，它的效率低。对视图操作比直接对表操作慢,可以用stored   procedure来代替她。特别的是不要用视图嵌套,嵌套视图增加了寻找原始资料的难度。我们看视图的本质：它是存放在服务器上的被优化好了的已经产生了查询规划的SQL。对单个表检索数据时，不要使用指向多个表的视图，直接从表检索或者仅仅包含这个表的视图上读，否则增加了不必要的开销,查询受到干扰.为了加快视图的查询，MsSQL增加了视图索引的功能。    
  
**34**、没有必要时不要用DISTINCT和ORDER   BY，这些动作可以改在客户端执行。它们增加了额外的开销。这同UNION   和UNION   ALL一样的道理。   SELECT   top   **20**   ad.companyname,comid,position,ad.referenceid,worklocation,   convert(varchar(**10**),ad.postDate,**120**)   as   postDate1,workyear,degreedescription   FROM   jobcn\_query.dbo.COMPANYAD\_query   ad   where   referenceID   in('JCNAD00329667','JCNAD132168','JCNAD00337748','JCNAD00338345','JCNAD00333138','JCNAD00303570',   'JCNAD00303569','JCNAD00303568','JCNAD00306698','JCNAD00231935','JCNAD00231933','JCNAD00254567',   'JCNAD00254585','JCNAD00254608','JCNAD00254607','JCNAD00258524','JCNAD00332133','JCNAD00268618',   'JCNAD00279196','JCNAD00268613')   order   by   postdate   desc    
  
**35**、在IN后面值的列表中，将出现最频繁的值放在最前面，出现得最少的放在最后面，减少判断的次数    
  
**36**、当用SELECT   INTO时，它会锁住系统表(sysobjects，sysindexes等等)，阻塞其他的连接的存取。创建临时表时用显示申明语句，而不是 select   INTO.   drop   table   t\_lxh   begin   tran   select   \*   into   t\_lxh   from   chineseresume   where   name   =   'XYZ'   --commit   在另一个连接中SELECT   \*   from   sysobjects可以看到   SELECT   INTO   会锁住系统表，Create   table   也会锁系统表(不管是临时表还是系统表)。所以千万不要在事物内使用它！！！这样的话如果是经常要用的临时表请使用实表，或者临时表变量。    
  
**37**、一般在GROUP   BY   个HAVING字句之前就能剔除多余的行，所以尽量不要用它们来做剔除行的工作。他们的执行顺序应该如下最优：select   的Where字句选择所有合适的行，Group   By用来分组个统计行，Having字句用来剔除多余的分组。这样Group   By   个Having的开销小，查询快.对于大的数据行进行分组和Having十分消耗资源。如果Group   BY的目的不包括计算，只是分组，那么用Distinct更快    
  
**38**、一次更新多条记录比分多次更新每次一条快,就是说批处理好    
  
**39**、少用临时表，尽量用结果集和Table类性的变量来代替它,Table   类型的变量比临时表好    
  
**40**、在SQL2000下，计算字段是可以索引的，需要满足的条件如下：    
  
  a、计算字段的表达是确定的    
  b、不能用在TEXT,Ntext，Image数据类型    
  c、必须配制如下选项   ANSI\_NULLS   =   ON,   ANSI\_PADDINGS   =   ON,   …….    
  
**41**、尽量将数据的处理工作放在服务器上，减少网络的开销，如使用存储过程。存储过程是编译好、优化过、并且被组织到一个执行规划里、且存储在数据库中的 SQL语句，是控制流语言的集合，速度当然快。反复执行的动态SQL,可以使用临时存储过程，该过程（临时表）被放在Tempdb中。以前由于SQL   SERVER对复杂的数学计算不支持，所以不得不将这个工作放在其他的层上而增加网络的开销。SQL2000支持UDFs,现在支持复杂的数学计算，函数的返回值不要太大，这样的开销很大。用户自定义函数象光标一样执行的消耗大量的资源，如果返回大的结果采用存储过程    
  
**42**、不要在一句话里再三的使用相同的函数，浪费资源,将结果放在变量里再调用更快    
  
**43**、SELECT   COUNT(\*)的效率教低，尽量变通他的写法，而EXISTS快.同时请注意区别：   select   count(Field   of   null)   from   Table   和   select   count(Field   of   NOT   null)   from   Table   的返回值是不同的。    
  
**44**、当服务器的内存够多时，配制线程数量   =   最大连接数+**5**，这样能发挥最大的效率；否则使用   配制线程数量 <最大连接数启用SQL   SERVER的线程池来解决,如果还是数量   =   最大连接数+**5**，严重的损害服务器的性能。    
  
**45**、按照一定的次序来访问你的表。如果你先锁住表A，再锁住表B，那么在所有的存储过程中都要按照这个顺序来锁定它们。如果你（不经意的）某个存储过程中先锁定表B，再锁定表A，这可能就会导致一个死锁。如果锁定顺序没有被预先详细的设计好，死锁很难被发现    
  
**46**、通过SQL   Server   Performance   Monitor监视相应硬件的负载   Memory:   Page   Faults   /   sec计数器如果该值偶尔走高，表明当时有线程竞争内存。如果持续很高，则内存可能是瓶颈。   Process:    
  
    **1**、%   DPC   Time   指在范例间隔期间处理器用在缓延程序调用(DPC)接收和提供服务的百分比。(DPC   正在运行的为比标准间隔优先权低的间隔)。   由于   DPC   是以特权模式执行的，DPC   时间的百分比为特权时间   百分比的一部分。这些时间单独计算并且不属于间隔计算总数的一部   分。这个总数显示了作为实例时间百分比的平均忙时。    
    **2**、%Processor   Time计数器　如果该参数值持续超过95%，表明瓶颈是CPU。可以考虑增加一个处理器或换一个更快的处理器。    
    **3**、%   Privileged   Time   指非闲置处理器时间用于特权模式的百分比。(特权模式是为**[操作系统](http://lib.csdn.net/base/operatingsystem" \t "_blank" \o "操作系统知识库)**组件和操纵硬件驱动程序而设计的一种处理模式。它允许直接访问硬件和所有内存。另一种模式为用户模式，它是一种为应用程序、环境分系统和整数分系统设计的一种有限处理模式。操作系统将应用程序线程转换成特权模式以访问操作系统服务)。   特权时间的   %   包括为间断和   DPC   提供服务的时间。特权时间比率高可能是由于失败设备产生的大数量的间隔而引起的。这个计数器将平均忙时作为样本时间的一部分显示。    
    **4**、%   User   Time表示耗费CPU的数据库操作，如排序，执行aggregate   functions等。如果该值很高，可考虑增加索引，尽量使用简单的表联接，水平分割大表格等方法来降低该值。   Physical   Disk:   Curretn   Disk   Queue   Length计数器该值应不超过磁盘数的1.**5**~2倍。要提高性能，可增加磁盘。   SQLServer:Cache   Hit   Ratio计数器该值越高越好。如果持续低于80%，应考虑增加内存。   注意该参数值是从SQL   Server启动后，就一直累加记数，所以运行经过一段时间后，该值将不能反映系统当前值。    
  
**47**、分析select   emp\_name   form   employee   where   salary   >   **3000**   在此语句中若salary是Float类型的，则优化器对其进行优化为Convert(float,**3000**)，因为3000是个整数，我们应在编程时使用3000.0而不要等运行时让DBMS进行转化。同样字符和整型数据的转换。