**数据库的优化问题**

**一、问题的提出**

**在应用系统开发初期，由于开发数据库数据比较少，对于查询SQL语句，复杂视图的的编写等体会不出SQL语句各种写法的性能优劣，但是如果将应用 系统提交实际应用后，随着数据库中数据的增加，系统的响应速度就成为目前系统需要解决的最主要的问题之一。系统优化中一个很重要的方面就是SQL语句的优 化。对于海量数据，劣质SQL语句和优质SQL语句之间的速度差别可以达到上百倍，可见对于一个系统不是简单地能实现其功能就可，而是要写出高质量的 SQL语句，提高系统的可用性。**

**在多数情况下，Oracle使用索引来更快地遍历表，优化器主要根据定义的索引来提高性能。但是，如果在SQL语句的where子句中写的 SQL代码不合理，就会造成优化器删去索引而使用全表扫描，一般就这种SQL语句就是所谓的劣质SQL语句。在编写SQL语句时我们应清楚优化器根据何种 原则来删除索引，这有助于写出高性能的SQL语句。**

**二、SQL语句编写注意问题**

**下面就某些SQL语句的where子句编写中需要注意的问题作详细介绍。在这些where子句中，即使某些列存在索引，但是由于编写了劣质的SQL，系统在运行该SQL语句时也不能使用该索引，而同样使用全表扫描，这就造成了响应速度的极大降低。**

**1. IS NULL 与 IS NOT NULL**

**不能用null作索引，任何包含null值的列都将不会被包含在索引中。即使索引有多列这样的情况下，只要这些列中有一列含有null，该列就会从索引中排除。也就是说如果某列存在空值，即使对该列建索引也不会提高性能。**

**任何在where子句中使用is null或is not null的语句优化器是不允许使用索引的。**

**2. 联接列**

**对于有联接的列，即使最后的联接值为一个静态值，优化器是不会使用索引的。我们一起来看一个例子，假定有一个职工表(employee)，对于 一个职工的姓和名分成两列存放(FIRST\_NAME和LAST\_NAME)，现在要查询一个叫比尔.克林顿(Bill Cliton)的职工。**

**下面是一个采用联接查询的SQL语句，**

**select \* from employss where first\_name||”||last\_name =’Beill Cliton';**

**上面这条语句完全可以查询出是否有Bill Cliton这个员工，但是这里需要注意，系统优化器对基于last\_name创建的索引没有使用。**

**当采用下面这种SQL语句的编写，Oracle系统就可以采用基于last\_name创建的索引。**

**\*\*\* where first\_name =’Beill’ and last\_name =’Cliton';**

**. 带通配符(%)的like语句 *词首使用通配符，索引不生效***

**同样以上面的例子来看这种情况。目前的需求是这样的，要求在职工表中查询名字中包含cliton的人。可以采用如下的查询SQL语句:**

**select \* from employee where last\_name like ‘%cliton%';**

**这里由于通配符(%)在搜寻词首出现，所以Oracle系统不使用last\_name的索引。在很多情况下可能无法避免这种情况，但是一定要心中有底，通 配符如此使用会降低查询速度。然而当通配符出现在字符串其他位置时，优化器就能利用索引。在下面的查询中索引得到了使用:**

**select \* from employee where last\_name like ‘c%';**

**4. Order by语句**

**ORDER BY语句决定了Oracle如何将返回的查询结果排序。Order by语句对要排序的列没有什么特别的限制，也可以将函数加入列中(象联接或者附加等)。任何在Order by语句的非索引项或者有计算表达式都将降低查询速度。**

**仔细检查order by语句以找出非索引项或者表达式，它们会降低性能。解决这个问题的办法就是重写order by语句以使用索引，也可以为所使用的列建立另外一个索引，同时应绝对避免在order by子句中使用表达式。**

**5. NOT**

**我们在查询时经常在where子句使用一些逻辑表达式，如大于、小于、等于以及不等于等等，也可以使用and(与)、or(或)以及not(非)。NOT可用来对任何逻辑运算符号取反。下面是一个NOT子句的例子:**

**… where not (status =’VALID’)**

**如果要使用NOT，则应在取反的短语前面加上括号，并在短语前面加上NOT运算符。NOT运算符包含在另外一个逻辑运算符中，这就是不等于(<>)运算符。换句话说，即使不在查询where子句中显式地加入NOT词，NOT仍在运算符中，见下例:**

**… where status <>’INVALID';**

**对这个查询，可以改写为不使用NOT:**

**select \* from employee where salary<3000 or salary>3000;**

**虽然这两种查询的结果一样，但是第二种查询方案会比第一种查询方案更快些。第二种查询允许Oracle对salary列使用索引，而第一种查询则不能使用索引。**

**虽然这两种查询的结果一样，但是第二种查询方案会比第一种查询方案更快些。第二种查询允许Oracle对salary列使用索引，而第一种查询则不能使用索引。**

**我们要做到不但会写SQL,还要做到写出性能优良的SQL,以下为笔者学习、摘录、并汇总部分资料与大家分享！**

**（1）      选择最有效率的表名顺序(只在基于规则的优化器中有效)：   
ORACLE 的解析器按照从右到左的顺序处理FROM子句中的表名，FROM子句中写在最后的表(基础表 driving table)将被最先处理，在FROM子句中包含多个表的情况下,你必须选择记录条数最少的表作为基础表。如果有3个以上的表连接查询, 那就需要选择交叉表(intersection table)作为基础表, 交叉表是指那个被其他表所引用的表.   
（2）      WHERE子句中的连接顺序．：   
ORACLE采用自下而上的顺序解析WHERE子句,根据这个原理,表之间的连接必须写在其他WHERE条件之前, 那些可以过滤掉最大数量记录的条件必须写在WHERE子句的末尾.   
（3）      SELECT子句中避免使用 ‘ \* ‘：   
ORACLE在解析的过程中, 会将’\*’ 依次转换成所有的列名, 这个工作是通过查询数据字典完成的, 这意味着将耗费更多的时间   
（4）      减少访问数据库的次数：   
ORACLE在内部执行了许多工作: 解析SQL语句, 估算索引的利用率, 绑定变量 , 读数据块等；   
（5）      在SQL\*Plus , SQL\*Forms和Pro\*C中重新设置ARRAYSIZE参数, 可以增加每次数据库访问的检索数据量 ,建议值为200   
（6）      使用DECODE函数来减少处理时间：   
使用DECODE函数可以避免重复扫描相同记录或重复连接相同的表.   
（7）      整合简单,无关联的数据库访问：   
如果你有几个简单的数据库查询语句,你可以把它们整合到一个查询中(即使它们之间没有关系)   
（8）      删除重复记录：   
最高效的删除重复记录方法 ( 因为使用了ROWID)例子：   
DELETE  FROM  EMP E  WHERE  E.ROWID > (SELECT MIN(X.ROWID)   
FROM  EMP X  WHERE  X.EMP\_NO = E.EMP\_NO);   
（9）      用TRUNCATE替代DELETE：   
当删除表中的记录时,在通常情况下, 回滚段(rollback segments ) 用来存放可以被恢复的信息. 如果你没有COMMIT事务,ORACLE会将数据恢复到删除之前的状态(准确地说是恢复到执行删除命令之前的状况) 而当运用TRUNCATE时, 回滚段不再存放任何可被恢复的信息.当命令运行后,数据不能被恢复.因此很少的资源被调用,执行时间也会很短. (译者按: TRUNCATE只在删除全表适用,TRUNCATE是DDL不是DML)   
（10） 尽量多使用COMMIT：   
只要有可能,在程序中尽量多使用COMMIT, 这样程序的性能得到提高,需求也会因为COMMIT所释放的资源而减少:   
COMMIT所释放的资源:   
a. 回滚段上用于恢复数据的信息.   
b. 被程序语句获得的锁   
c. redo log buffer 中的空间   
d. ORACLE为管理上述3种资源中的内部花费   
（11） 用Where子句替换HAVING子句：   
避免使用HAVING子句, HAVING 只会在检索出所有记录之后才对结果集进行过滤. 这个处理需要排序,总计等操作. 如果能通过WHERE子句限制记录的数目,那就能减少这方面的开销. (非oracle中)on、where、having这三个都可以加条件的子句中，on是最先执行，where次之，having最后，因为on是先把不 符合条件的记录过滤后才进行统计，它就可以减少中间运算要处理的数据，按理说应该速度是最快的，where也应该比having快点的，因为它过滤数据后 才进行sum，在两个表联接时才用on的，所以在一个表的时候，就剩下where跟having比较了。在这单表查询统计的情况下，如果要过滤的条件没有涉及到要计算字段，那它们的结果是一样的，只是where可以使用rushmore技术，而having就不能，在速度上后者要慢如果要涉及到计算的字 段，就表示在没计算之前，这个字段的值是不确定的，根据上篇写的工作流程，where的作用时间是在计算之前就完成的，而having就是在计算后才起作 用的，所以在这种情况下，两者的结果会不同。在多表联接查询时，on比where更早起作用。系统首先根据各个表之间的联接条件，把多个表合成一个临时表 后，再由where进行过滤，然后再计算，计算完后再由having进行过滤。由此可见，要想过滤条件起到正确的作用，首先要明白这个条件应该在什么时候起作用，然后再决定放在那里   
（12） 减少对表的查询：   
在含有子查询的SQL语句中,要特别注意减少对表的查询.例子：   
    SELECT  TAB\_NAME FROM TABLES WHERE (TAB\_NAME,DB\_VER) = ( SELECT   
TAB\_NAME,DB\_VER FROM  TAB\_COLUMNS  WHERE  VERSION = 604)   
（13） 通过内部函数提高SQL效率.：   
复杂的SQL往往牺牲了执行效率. 能够掌握上面的运用函数解决问题的方法在实际工作中是非常有意义的   
（14） 使用表的别名(Alias)：   
当在SQL语句中连接多个表时, 请使用表的别名并把别名前缀于每个Column上.这样一来,就可以减少解析的时间并减少那些由Column歧义引起的语法错误.   
（15） 用EXISTS替代IN、用NOT EXISTS替代NOT IN：   
在许多基于基础表的查询中,为了满足一个条件,往往需要对另一个表进行联接.在这种情况下, 使用EXISTS(或NOT EXISTS)通常将提高查询的效率. 在子查询中,NOT IN子句将执行一个内部的排序和合并. 无论在哪种情况下,NOT IN都是最低效的 (因为它对子查询中的表执行了一个全表遍历). 为了避免使用NOT IN ,我们可以把它改写成外连接(Outer Joins)或NOT EXISTS.   
例子：   
（高效）SELECT \* FROM  EMP (基础表)  WHERE  EMPNO > 0  AND  EXISTS (SELECT ‘X’  FROM DEPT  WHERE  DEPT.DEPTNO = EMP.DEPTNO  AND  LOC = ‘MELB’)   
(低效)SELECT  \* FROM  EMP (基础表)  WHERE  EMPNO > 0  AND  DEPTNO IN(SELECT DEPTNO  FROM  DEPT  WHERE  LOC = ‘MELB’)   
（16） 识别’低效执行’的SQL语句：   
虽然目前各种关于SQL优化的图形化工具层出不穷,但是写出自己的SQL工具来解决问题始终是一个最好的方法：   
SELECT  EXECUTIONS , DISK\_READS, BUFFER\_GETS,   
ROUND((BUFFER\_GETS-DISK\_READS)/BUFFER\_GETS,2) Hit\_radio,   
ROUND(DISK\_READS/EXECUTIONS,2) Reads\_per\_run,   
SQL\_TEXT   
FROM  V$SQLAREA   
WHERE  EXECUTIONS>0   
AND  BUFFER\_GETS > 0   
AND  (BUFFER\_GETS-DISK\_READS)/BUFFER\_GETS < 0.8   
ORDER BY  4 DESC;  
  
（17） 用索引提高效率：   
索引是表的一个概念部分,用来提高检索数据的效率，ORACLE使用了一个复杂的自平衡B-tree结构. 通常,通过索引查询数据比全表扫描要快. 当ORACLE找出执行查询和Update语句的最佳路径时, ORACLE优化器将使用索引. 同样在联结多个表时使用索引也可以提高效率. 另一个使用索引的好处是,它提供了主键(primary key)的唯一性验证.。那些LONG或LONG RAW数据类型, 你可以索引几乎所有的列. 通常, 在大型表中使用索引特别有效. 当然,你也会发现, 在扫描小表时,使用索引同样能提高效率. 虽然使用索引能得到查询效率的提高,但是我们也必须注意到它的代价. 索引需要空间来存储,也需要定期维护, 每当有记录在表中增减或索引列被修改时, 索引本身也会被修改. 这意味着每条记录的INSERT , DELETE , UPDATE将为此多付出4 , 5 次的磁盘I/O . 因为索引需要额外的存储空间和处理,那些不必要的索引反而会使查询反应时间变慢.。定期的重构索引是有必要的.：   
ALTER  INDEX <INDEXNAME> REBUILD <TABLESPACENAME>   
18） 用EXISTS替换DISTINCT：   
当提交一个包含一对多表信息(比如部门表和雇员表)的查询时,避免在SELECT子句中使用DISTINCT. 一般可以考虑用EXIST替换, EXISTS 使查询更为迅速,因为RDBMS核心模块将在子查询的条件一旦满足后,立刻返回结果. 例子：   
      (低效):   
SELECT  DISTINCT  DEPT\_NO,DEPT\_NAME  FROM  DEPT D , EMP E   
WHERE  D.DEPT\_NO = E.DEPT\_NO   
(高效):   
SELECT  DEPT\_NO,DEPT\_NAME  FROM  DEPT D  WHERE  EXISTS ( SELECT ‘X’   
FROM  EMP E  WHERE E.DEPT\_NO = D.DEPT\_NO);   
（19） sql语句用大写的；因为oracle总是先解析sql语句，把小写的字母转换成大写的再执行   
（20） 在java代码中尽量少用连接符“＋”连接字符串！   
（21） 避免在索引列上使用NOT 通常，　   
我们要避免在索引列上使用NOT, NOT会产生在和在索引列上使用函数相同的影响. 当ORACLE”遇到”NOT,他就会停止使用索引转而执行全表扫描.   
（22） 避免在索引列上使用计算．   
WHERE子句中，如果索引列是函数的一部分．优化器将不使用索引而使用全表扫描．   
举例:   
低效：   
SELECT … FROM  DEPT  WHERE SAL \* 12 > 25000;   
高效:   
SELECT … FROM DEPT WHERE SAL > 25000/12;   
（23） 用>=替代>   
高效:   
SELECT \* FROM  EMP  WHERE  DEPTNO >=4   
低效:   
SELECT \* FROM EMP WHERE DEPTNO >3   
两者的区别在于, 前者DBMS将直接跳到第一个DEPT等于4的记录而后者将首先定位到DEPTNO=3的记录并且向前扫描到第一个DEPT大于3的记录.   
（24） 用UNION替换OR (适用于索引列)   
通常情况下, 用UNION替换WHERE子句中的OR将会起到较好的效果. 对索引列使用OR将造成全表扫描. 注意, 以上规则只针对多个索引列有效. 如果有column没有被索引, 查询效率可能会因为你没有选择OR而降低. 在下面的例子中, LOC\_ID 和REGION上都建有索引.   
高效:   
SELECT LOC\_ID , LOC\_DESC , REGION   
FROM LOCATION   
WHERE LOC\_ID = 10   
UNION   
SELECT LOC\_ID , LOC\_DESC , REGION   
FROM LOCATION   
WHERE REGION = “MELBOURNE”   
低效:   
SELECT LOC\_ID , LOC\_DESC , REGION   
FROM LOCATION   
WHERE LOC\_ID = 10 OR REGION = “MELBOURNE”   
如果你坚持要用OR, 那就需要返回记录最少的索引列写在最前面.   
（25） 用IN来替换OR    
这是一条简单易记的规则，但是实际的执行效果还须检验，在ORACLE8i下，两者的执行路径似乎是相同的．　   
低效:   
SELECT…. FROM LOCATION WHERE LOC\_ID = 10 OR LOC\_ID = 20 OR LOC\_ID = 30   
高效   
SELECT… FROM LOCATION WHERE LOC\_IN  IN (10,20,30);   
（26） 避免在索引列上使用IS NULL和IS NOT NULL   
避免在索引中使用任何可以为空的列，ORACLE将无法使用该索引．对于单列索引，如果列包含空值，索引中将不存在此记录. 对于复合索引，如果每个列都为空，索引中同样不存在此记录.　如果至少有一个列不为空，则记录存在于索引中．举例: 如果唯一性索引建立在表的A列和B列上, 并且表中存在一条记录的A,B值为(123,null) , ORACLE将不接受下一条具有相同A,B值（123,null）的记录(插入). 然而如果所有的索引列都为空，ORACLE将认为整个键值为空而空不等于空. 因此你可以插入1000 条具有相同键值的记录,当然它们都是空! 因为空值不存在于索引列中,所以WHERE子句中对索引列进行空值比较将使ORACLE停用该索引.   
低效: (索引失效)   
SELECT … FROM  DEPARTMENT  WHERE  DEPT\_CODE IS NOT NULL;   
高效: (索引有效)   
SELECT … FROM  DEPARTMENT  WHERE  DEPT\_CODE >=0;   
（27） 总是使用索引的第一个列：   
如果索引是建立在多个列上, 只有在它的第一个列(leading column)被where子句引用时,优化器才会选择使用该索引. 这也是一条简单而重要的规则，当仅引用索引的第二个列时,优化器使用了全表扫描而忽略了索引   
28） 用UNION-ALL 替换UNION ( 如果有可能的话)：   
当SQL 语句需要UNION两个查询结果集合时,这两个结果集合会以UNION-ALL的方式被合并, 然后在输出最终结果前进行排序. 如果用UNION ALL替代UNION, 这样排序就不是必要了. 效率就会因此得到提高. 需要注意的是，UNION ALL 将重复输出两个结果集合中相同记录. 因此各位还是要从业务需求分析使用UNION ALL的可行性. UNION 将对结果集合排序,这个操作会使用到SORT\_AREA\_SIZE这块内存. 对于这块内存的优化也是相当重要的. 下面的SQL可以用来查询排序的消耗量   
低效：   
SELECT  ACCT\_NUM, BALANCE\_AMT   
FROM  DEBIT\_TRANSACTIONS   
WHERE TRAN\_DATE = ’31-DEC-95′   
UNION   
SELECT ACCT\_NUM, BALANCE\_AMT   
FROM DEBIT\_TRANSACTIONS   
WHERE TRAN\_DATE = ’31-DEC-95′   
高效:   
SELECT ACCT\_NUM, BALANCE\_AMT   
FROM DEBIT\_TRANSACTIONS   
WHERE TRAN\_DATE = ’31-DEC-95′   
UNION ALL   
SELECT ACCT\_NUM, BALANCE\_AMT   
FROM DEBIT\_TRANSACTIONS   
WHERE TRAN\_DATE = ’31-DEC-95′   
（29） 用WHERE替代ORDER BY：   
ORDER BY 子句只在两种严格的条件下使用索引.   
ORDER BY中所有的列必须包含在相同的索引中并保持在索引中的排列顺序.   
ORDER BY中所有的列必须定义为非空.   
WHERE子句使用的索引和ORDER BY子句中所使用的索引不能并列.   
例如:   
表DEPT包含以下列:   
DEPT\_CODE PK NOT NULL   
DEPT\_DESC NOT NULL   
DEPT\_TYPE NULL   
低效: (索引不被使用)   
SELECT DEPT\_CODE FROM  DEPT  ORDER BY  DEPT\_TYPE   
高效: (使用索引)   
SELECT DEPT\_CODE  FROM  DEPT  WHERE  DEPT\_TYPE > 0   
（30） 避免改变索引列的类型.:   
当比较不同数据类型的数据时, ORACLE自动对列进行简单的类型转换.   
假设 EMPNO是一个数值类型的索引列.   
SELECT …  FROM EMP  WHERE  EMPNO = ‘123′   
实际上,经过ORACLE类型转换, 语句转化为:   
SELECT …  FROM EMP  WHERE  EMPNO = TO\_NUMBER(‘123′)   
幸运的是,类型转换没有发生在索引列上,索引的用途没有被改变.   
现在,假设EMP\_TYPE是一个字符类型的索引列.   
SELECT …  FROM EMP  WHERE EMP\_TYPE = 123   
这个语句被ORACLE转换为:   
SELECT …  FROM EMP  WHERETO\_NUMBER(EMP\_TYPE)=123   
因为内部发生的类型转换, 这个索引将不会被用到! 为了避免ORACLE对你的SQL进行隐式的类型转换, 最好把类型转换用显式表现出来. 注意当字符和数值比较时, ORACLE会优先转换数值类型到字符类型   
（31） 需要当心的WHERE子句:   
某些SELECT 语句中的WHERE子句不使用索引. 这里有一些例子.   
在下面的例子里, (1)‘!=’ 将不使用索引. 记住, 索引只能告诉你什么存在于表中, 而不能告诉你什么不存在于表中. (2) ‘ ¦ ¦’是字符连接函数. 就象其他函数那样, 停用了索引. (3) ‘+’是数学函数. 就象其他数学函数那样, 停用了索引. (4)相同的索引列不能互相比较,这将会启用全表扫描.   
（32） a. 如果检索数据量超过30%的表中记录数.使用索引将没有显著的效率提高.   
b. 在特定情况下, 使用索引也许会比全表扫描慢, 但这是同一个数量级上的区别. 而通常情况下,使用索引比全表扫描要块几倍乃至几千倍!   
（33） 避免使用耗费资源的操作:   
带有DISTINCT,UNION,MINUS,INTERSECT,ORDER BY的SQL语句会启动SQL引擎   
执行耗费资源的排序(SORT)功能. DISTINCT需要一次排序操作, 而其他的至少需要执行两次排序. 通常, 带有UNION, MINUS , INTERSECT的SQL语句都可以用其他方式重写. 如果你的数据库的SORT\_AREA\_SIZE调配得好, 使用UNION , MINUS, INTERSECT也是可以考虑的, 毕竟它们的可读性很强   
（34） 优化GROUP BY:   
提高GROUP BY 语句的效率, 可以通过将不需要的记录在GROUP BY 之前过滤掉.下面两个查询返回相同结果但第二个明显就快了许多.   
低效:   
SELECT JOB , AVG(SAL)   
FROM EMP   
GROUP by JOB   
HAVING JOB = ‘PRESIDENT’   
OR JOB = ‘MANAGER’   
高效:   
SELECT JOB , AVG(SAL)   
FROM EMP   
WHERE JOB = ‘PRESIDENT’   
OR JOB = ‘MANAGER’   
GROUP by JOB   
  
====================================  
====================================  
如果你正在负责一个基于SQL Server的项目，或者你刚刚接触SQL Server，你都有可能要面临一些数据库性能的问题，这篇文章会为你提供一些有用的指导（其中大多数也可以用于其它的DBMS）。   
在这里，我不打算介绍使用SQL Server的窍门，也不能提供一个包治百病的方案，我所做的是总结一些经验—-关于如何形成一个好的设计。这些经验来自我过去几年中经受的教训，一直来，我看到许多同样的设计错误被一次又一次的重复。   
一、了解你用的工具   
不要轻视这一点，这是我在这篇文章中讲述的最关键的一条。也许你也看到有很多的SQL Server程序员没有掌握全部的T-SQL命令和SQL Server提供的那些有用的工具。   
“什么？我要浪费一个月的时间来学习那些我永远也不会用到的SQL命令？？？”，你也许会这样说。对的，你不需要这样做。但是你应该用一个周末浏览所有的 T-SQL命令。在这里，你的任务是了解，将来，当你设计一个查询时，你会记起来：“对了，这里有一个命令可以完全实现我需要的功能”，于是，到MSDN 查看这个命令的确切语法。   
二、不要使用游标   
让我再重复一遍：不要使用游标。如果你想破坏整个系统的性能的话，它们倒是你最有效的首选办法。大多数的初学者都使用游标，而没有意识到它们对性能造成的影响。它们占用内存，还用它们那些不可思议的方式锁定表，另外，它们简直就像蜗牛。而最糟糕的是，它们可以使你的DBA所能做的一切性能优化等于没做。不 知你是否知道每执行一次FETCH就等于执行一次SELECT命令？这意味着如果你的游标有10000条记录，它将执行10000次SELECT！如果你 使用一组SELECT、UPDATE或者DELETE来完成相应的工作，那将有效率的多。   
初学者一般认为使用游标是一种比较熟悉和舒适的编程方式，可很不幸，这会导致糟糕的性能。显然，SQL的总体目的是你要实现什么，而不是怎样实现。   
我曾经用T-SQL重写了一个基于游标的存储过程，那个表只有100,000条记录，原来的存储过程用了40分钟才执行完毕，而新的存储过程只用了10秒钟。在这里，我想你应该可以看到一个不称职的程序员究竟在干了什么！！！   
我们可以写一个小程序来取得和处理数据并且更新数据库，这样做有时会更有效。记住：对于循环，T-SQL无能为力。   
我再重新提醒一下：使用游标没有好处。除了DBA的工作外，我从来没有看到过使用游标可以有效的完成任何工作。   
三、规范化你的数据表   
为什么不规范化数据库？大概有两个借口：出于性能的考虑和纯粹因为懒惰。至于第二点，你迟早得为此付出代价。而关于性能的问题，你不需要优化根本就不慢的东西。我经常看到一些程序员“反规范化”数据库，他们的理由是“原来的设计太慢了”，可结果却常常是他们让系统更慢了。DBMS被设计用来处理规范数据库 的，因此，记住：按照规范化的要求设计数据库。   
四、不要使用SELECT \*   
这点不太容易做到，我太了解了，因为我自己就经常这样干。可是，如果在SELECT中指定你所需要的列，那将会带来以下的好处：   
1 减少内存耗费和网络的带宽   
2 你可以得到更安全的设计   
3 给查询优化器机会从索引读取所有需要的列   
五、了解你将要对数据进行的操作   
为你的数据库创建一个健壮的索引，那可是功德一件。可要做到这一点简直就是一门艺术。每当你为一个表添加一个索引，SELECT会更快了，可INSERT 和DELETE却大大的变慢了，因为创建了维护索引需要许多额外的工作。显然，这里问题的关键是：你要对这张表进行什么样的操作。这个问题不太好把握，特别是涉及DELETE和UPDATE时，因为这些语句经常在WHERE部分包含SELECT命令。   
六、不要给“性别”列创建索引   
首先，我们必须了解索引是如何加速对表的访问的。你可以将索引理解为基于一定的标准上对表进行划分的一种方式。如果你给类似于“性别”这样的列创建了一个 索引，你仅仅是将表划分为两部分：男和女。你在处理一个有1,000,000条记录的表，这样的划分有什么意义？记住：维护索引是比较费时的。当你设计索 引时，请遵循这样的规则：根据列可能包含不同内容的数目从多到少排列，比如：姓名+省份+性别。   
七、使用事务   
请使用事务，特别是当查询比较耗时。如果系统出现问题，这样做会救你一命的。一般有些经验的程序员都有体会—–你经常会碰到一些不可预料的情况会导致存储过程崩溃。   
八、小心死锁   
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，死锁是不太容易被发现的。   
九、不要打开大的数据集   
一个经常被提出的问题是：我怎样才能迅速的将100000条记录添加到ComboBox中？这是不对的，你不能也不需要这样做。很简单，你的用户要浏览 100000条记录才能找到需要的记录，他一定会诅咒你的。在这里，你需要的是一个更好的UI，你需要为你的用户显示不超过100或200条记录。   
十、不要使用服务器端游标   
与服务器端游标比起来，客户端游标可以减少服务器和网络的系统开销，并且还减少锁定时间。   
十一、使用参数查询   
有时，我在CSDN技术论坛看到类似这样的问题：“SELECT \* FROM a WHERE a.id=’A’B，因为单引号查询发生异常，我该怎么办？”，而普遍的回答是：用两个单引号代替单引号。这是错误的。这样治标不治本，因为你还会在其他 一些字符上遇到这样的问题，更何况这样会导致严重的bug，除此以外，这样做还会使SQL Server的缓冲系统无法发挥应有的作用。使用参数查询，釜底抽薪，这些问题统统不存在了。   
十二、在程序编码时使用大数据量的数据库   
程序员在开发中使用的测试数据库一般数据量都不大，可经常的是最终用户的数据量都很大。我们通常的做法是不对的，原因很简单：现在硬盘不是很贵，可为什么性能问题却要等到已经无可挽回的时候才被注意呢？   
十三、不要使用INSERT导入大批的数据   
请不要这样做，除非那是必须的。使用UTS或者BCP，这样你可以一举而兼得灵活性和速度。   
十四、注意超时问题   
查询数据库时，一般数据库的缺省都比较小，比如15秒或者30秒。而有些查询运行时间要比这长，特别是当数据库的数据量不断变大时。   
十五、不要忽略同时修改同一记录的问题   
有时候，两个用户会同时修改同一记录，这样，后一个修改者修改了前一个修改者的操作，某些更新就会丢失。处理这种情况不是很难：创建一个timestamp字段，在写入前检查它，如果允许，就合并修改，如果存在冲突，提示用户。   
十六、在细节表中插入纪录时，不要在主表执行SELECT MAX(ID)   
这是一个普遍的错误，当两个用户在同一时间插入数据时，这会导致错误。你可以使用SCOPE\_IDENTITY，IDENT\_CURRENT和IDENTITY。如果可能，不要使用IDENTITY，因为在有触发器的情况下，它会引起一些问题（详见这里的讨论）。   
十七、避免将列设为NULLable   
如果可能的话，你应该避免将列设为NULLable。系统会为NULLable列的每一行分配一个额外的字节，查询时会带来更多的系统开销。另外，将列设为NULLable使编码变得复杂，因为每一次访问这些列时都必须先进行检查。   
我并不是说NULLS是麻烦的根源，尽管有些人这样认为。我认为如果你的业务规则中允许“空数据”，那么，将列设为NULLable有时会发挥很好的作用，但是，如果在类似下面的情况中使用NULLable，那简直就是自讨苦吃。   
CustomerName1   
CustomerAddress1   
CustomerEmail1   
CustomerName2   
CustomerAddress2   
CustomerEmail3   
CustomerName1   
CustomerAddress2   
CustomerEmail3   
如果出现这种情况，你需要规范化你的表了。   
十八、尽量不要使用TEXT数据类型   
除非你使用TEXT处理一个很大的数据，否则不要使用它。因为它不易于查询，速度慢，用的不好还会浪费大量的空间。一般的，VARCHAR可以更好的处理你的数据。   
十九、尽量不要使用临时表   
尽量不要使用临时表，除非你必须这样做。一般使用子查询可以代替临时表。使用临时表会带来系统开销，如果你是用COM+进行编程，它还会给你带来很大的麻 烦，因为COM+使用数据库连接池而临时表却自始至终都存在。SQL Server提供了一些替代方案，比如Table数据类型。   
二十、学会分析查询   
SQL Server查询分析器是你的好伙伴，通过它你可以了解查询和索引是如何影响性能的。   
二十一、使用参照完整性   
定义主健、唯一性约束和外键，这样做可以节约大量的时间。**

**在条件有限的条件下，我们可以调整应用程序的SQL质量:**

**1. 不要进行全表扫描(Full Table Scan):全表扫描导致大量的I/O**

**2. 尽量建好和使用好索引:建索引也是有讲究的，在建索引时，也不是索引越多越好，当一个表的索引达到4个以上时，ORACLE的性能可能还是改善不了，因为 OLTP系统每表超过5个索引即会降低性能，而且在一个sql 中， Oracle 从不能使用超过 5个索引;当我们用到GROUP BY和ORDER BY时,ORACLE就会自动对数据进行排序,而ORACLE在INIT.ORA中决定了sort\_area\_size区的大小,当排序不能在我们给定的 排序区完成时,ORACLE就会在磁盘中进行排序,也就是我们讲的临时表空间中排序, 过多的磁盘排序将会令 free buffer waits 的值变高,而这个区间并不只是用于排序的,对于开发人员我提出如下忠告:**

**1)、select,update,delete 语句中的子查询应当有规律地查找少于20%的表行.如果一个语句查找的行数超过总行数的20%,它将不能通过使用索引获得性能上的提高.**

**2)、索引可能产生碎片,因为记录从表中删除时,相应也从表的索引中删除.表释放的空间可以再用,而索引释放的空间却不能再用.频繁进行删除操 作的被索引的表,应当阶段性地重建索引,以避免在索引中造成空间碎片,影响性能.在许可的条件下,也可以阶段性地truncate表,truncate命 令删除表中所有记录,也删除索引碎片.**

**3)、在使用索引时一定要按索引对应字段的顺序进行引用。**

**4)、用(+)比用NOT IN更有效率。**

**降低ORACLE的竞争:**

**先讲几个ORACLE的几个参数，这几个参数关系到ORACLE的竞争:**

**1)、freelists 和 freelist 组:他们负责ORACLE的处理表和索引的空间管理;**

**2)、pctfree 及 pctused:该参数决定了freelists 和 freelist 组的行为，pctfree 和pctused 参数的唯一目的就是为了控制块如何在 freelists 中进出**

**设置好pctfree 及 pctused对块在freelists的移走和读取很重要。**

**其他参数的设置**

**1)、包括SGA区(系统全局区):系统全局区(SGA)是一个分配给Oracle 的包含一个 Oracle 实例的数据库的控制信息内存段。**

**主要包括数据库高速缓存(the database buffer cache)，**

**重演日志缓存(the redo log buffer)，**

**共享池(the shared pool)，**

**数据字典缓存(the data dictionary cache)以及其它各方面的信息**

**2)、db\_block\_buffers(数据高速缓冲区)访问过的数据都放在这一片内存区域，该参数越大，Oracle在内存中找到相同数据的可能性就越大，也即加快了查询速度。**

**3)、share\_pool\_size (SQL共享缓冲池):该参数是库高速缓存和数据字典的高速缓存。**

**4)、Log\_buffer (重演日志缓冲区)**

**5)、sort\_area\_size(排序区)**

**6)、processes (同时连接的进程数)**

**7)、db\_block\_size (数据库块大小):Oracle默认块为2KB，太小了，因为如果我们有一个8KB的数据，则2KB块的数据库要读4次盘，才能读完，而8KB块的数据库 只要1次就读完了，大大减少了I/O操作。数据库安装完成后，就不能再改变db\_block\_size的值了，只能重新建立数据库并且建库时，要选择手工 安装数据库。**

**8)、open\_links (同时打开的链接数)**

**9)、dml\_locks**

**10)、open\_cursors (打开光标数)**

**11)、dbwr\_io\_slaves (后台写进程数)**

**6. IN和EXISTS**

**有时候会将一列和一系列值相比较。最简单的办法就是在where子句中使用子查询。在where子句中可以使用两种格式的子查询。**

**第一种格式是使用IN操作符:**

**… where column in(select \* from … where …);**

**第二种格式是使用EXIST操作符:**

**… where exists (select ‘X’ from …where …);**

**常见优化Sql查询性能的方法有哪些？**

**1，什么是“执行计划”？**

**执行计划是数据库根据SQL语句和相关表的统计信息作出的一个查询方案，这个方案是由查询优化器自动分析产生的，比如一条SQL语句如果用来从一个 10万条记录的表中查1条记录，那查询优化器会选择“索引查找”方式，如果该表进行了归档，当前只剩下5000条记录了，那查询优化器就会改变方案，采用 “全表扫描”方式。  
可见，执行计划并不是固定的，它是“带有相当个性的”。如何产生一个正确的执行计划呢？   
(1) SQL语句是否清晰地告诉查询优化器它想干什么？   
(2) 查询优化器得到的数据库统计信息是否是最新的、正确的？**

**2、 如何写出统一SQL语句**

**对于以下两句SQL语句，很多人人认为是相同的，但是，数据库查询优化器认为是不同的。**

* **select \* from dual**
* **select \* From dual**

**虽然只是大小写不同，查询分析器就认为是两句不同的SQL语句，必须进行两次解析。生成2个执行计划。所以作为程序员，应该保证相同的查询语句在任何地方都一致，多一个空格都不行！**

**3、 不要把SQL语句写得太长，太过冗余**

**一般，将一个Select语句的结果作为子集，然后从该子集中再进行查询，这种一层嵌套语句还是比较常见的，但是根据经验，超过3层嵌套，查询优化器就很容易给出错误的执行计划。因为它被绕晕了。像这种类似人工智能的东西，终究比人的分辨力要差些，如果人都看晕了，我可以保证数据库也会晕的。**

**另外，执行计划是可以被重用的，越简单的SQL语句被重用的可能性越高。而复杂的SQL语句只要有一个字符发生变化就必须重新解析，然后再把这一大堆垃圾塞在内存里。可想而知，数据库的效率会何等低下。**

**4、考虑使用“临时表”暂存中间结果**

**简化SQL语句的重要方法就是采用临时表暂存中间结果，但是，临时表的好处远远不止这些，将临时结果暂存在临时表，后面的查询就在tempdb中了，这可以避免程序中多次扫描主表，也大大减少了程序执行中“共享锁”阻塞“更新锁”，减少了阻塞，提高了并发性能。**

**5、 OLTP系统SQL语句必须采用绑定变量**

* **select \* from orderheader where changetime >’2010-10-20 00:00:01′**
* **select \* from orderheader where changetime >’2010-09-22 00:00:01′**

**以上两句语句，查询优化器认为是不同的SQL语句，需要解析两次。如果采用绑定变量**

* **select\*from orderheader where changetime >@chgtime**

**@chgtime变量可以传入任何值，这样大量的类似查询可以重用该执行计划了，可以大大降低数据库解析SQL语句的负担。一次解析，多次重用，是提高数据库效率的原则。**

**6、 绑定变量窥测**

**事物都存在两面性，绑定变量对大多数OLTP处理是适用的，但是也有例外。比如在where条件中的字段是“倾斜字段”的时候。**

**“倾斜字段”指该列中的绝大多数的值都是相同的，比如一张人口调查表，其中“民族”这列，90%以上都是汉族。那么如果一个SQL语句要查询30岁的汉族人口有多少，那“民族”这列必然要被放在where条件中。这个时候如果采用绑定变量@nation会存在很大问题。**

**试想如果@nation传入的第一个值是“汉族”，那整个执行计划必然会选择表扫描。然后，第二个值传入的是“布依族”，按理说“布依族”占的比例可能只有万分之一，应该采用索引查找。但是，由于重用了第一次解析的“汉族”的那个执行计划，那么第二次也将采用表扫描方式。这个问题就是著名的“绑定变量窥测”，建议对于“倾斜字段”不要采用绑定变量。**

**7、 只在必要的情况下才使用begin tran**

**SQL Server中一句SQL语句默认就是一个事务，在该语句执行完成后也是默认commit的。其实，这就是begin tran的一个最小化的形式，好比在每句语句开头隐含了一个begin tran，结束时隐含了一个commit。**

**有些情况下，我们需要显式声明begin tran，比如做“插、删、改”操作需要同时修改几个表，要求要么几个表都修改成功，要么都不成功。begin tran 可以起到这样的作用，它可以把若干SQL语句套在一起执行，最后再一起commit。 好处是保证了数据的一致性，但任何事情都不是完美无缺的。Begin tran付出的代价是在提交之前，所有SQL语句锁住的资源都不能释放，直到commit掉。**

**可见，如果Begin tran套住的SQL语句太多，那数据库的性能就糟糕了。在该大事务提交之前，必然会阻塞别的语句，造成block很多。**

**Begin tran使用的原则是，在保证数据一致性的前提下，begin tran 套住的SQL语句越少越好！有些情况下可以采用触发器同步数据，不一定要用begin tran。**

**8、 部分SQL查询语句加上nolock**

**在SQL语句中加nolock是提高SQL Server并发性能的重要手段，在oracle中并不需要这样做，因为oracle的结构更为合理，有undo表空间保存“数据前影”，该数据如果在修改中还未commit，那么你读到的是它修改之前的副本，该副本放在undo表空间中。这样，oracle的读、写可以做到互不影响，这也是oracle 广受称赞的地方。SQL Server 的读、写是会相互阻塞的，为了提高并发性能，对于一些查询，可以加上nolock，这样读的时候可以允许写，但缺点是可能读到未提交的脏数据。使用 nolock有3条原则。**

1. **查询的结果用于“插、删、改”的不能加nolock ！**
2. **查询的表属于频繁发生页分裂的，慎用nolock ！**
3. **使用临时表一样可以保存“数据前影”，起到类似oracle的undo表空间的功能，能采用临时表提高并发性能的，不要用nolock 。**

**9、 聚集索引没有建在表的顺序字段上，该表容易发生页分裂**

**比如订单表，有订单编号orderid，也有客户编号contactid，那么聚集索引应该加在哪个字段上呢？对于该表，订单编号是顺序添加的，如果在orderid上加聚集索引，新增的行都是添加在末尾，这样不容易经常产生页分裂。然而，由于大多数查询都是根据客户编号来查的，因此，将聚集索引加在contactid上才有意义。而contactid对于订单表而言，并非顺序字段。**

**比如“张三”的“contactid”是001，那么“张三”的订单信息必须都放在这张表的第一个数据页上，如果今天“张三”新下了一个订单，那该订单信息不能放在表的最后一页，而是第一页！如果第一页放满了呢？很抱歉，该表所有数据都要往后移动为这条记录腾地方。**

**SQL Server的索引和Oracle的索引是不同的，SQL Server的聚集索引实际上是对表按照聚集索引字段的顺序进行了排序，相当于oracle的索引组织表。SQL Server的聚集索引就是表本身的一种组织形式，所以它的效率是非常高的。也正因为此，插入一条记录，它的位置不是随便放的，而是要按照顺序放在该放的数据页，如果那个数据页没有空间了，就引起了页分裂。所以很显然，聚集索引没有建在表的顺序字段上，该表容易发生页分裂。**

**曾经碰到过一个情况，一位哥们的某张表重建索引后，插入的效率大幅下降了。估计情况大概是这样的。该表的聚集索引可能没有建在表的顺序字段上，该表经常被归档，所以该表的数据是以一种稀疏状态存在的。比如张三下过20张订单，而最近3个月的订单只有5张，归档策略是保留3个月数据，那么张三过去的 15张订单已经被归档，留下15个空位，可以在insert发生时重新被利用。在这种情况下由于有空位可以利用，就不会发生页分裂。但是查询性能会比较低，因为查询时必须扫描那些没有数据的空位。**

**重建聚集索引后情况改变了，因为重建聚集索引就是把表中的数据重新排列一遍，原来的空位没有了，而页的填充率又很高，插入数据经常要发生页分裂，所以性能大幅下降。**

**对于聚集索引没有建在顺序字段上的表，是否要给与比较低的页填充率？是否要避免重建聚集索引？是一个值得考虑的问题！**

**10、加nolock后查询经常发生页分裂的表，容易产生跳读或重复读**

**加nolock后可以在“插、删、改”的同时进行查询，但是由于同时发生“插、删、改”，在某些情况下，一旦该数据页满了，那么页分裂不可避免，而此时nolock的查询正在发生，比如在第100页已经读过的记录，可能会因为页分裂而分到第101页，这有可能使得nolock查询在读101页时重复读到该条数据，产生“重复读”。同理，如果在100页上的数据还没被读到就分到99页去了，那nolock查询有可能会漏过该记录，产生“跳读”。**

**上面提到的哥们，在加了nolock后一些操作出现报错，估计有可能因为nolock查询产生了重复读，2条相同的记录去插入别的表，当然会发生主键冲突。**

**11、合理使用like模糊查询**

**有的时候会需要进行一些模糊查询比如：**

* **select \* from contact where username like ‘%yue%’**

**关键词 %yue%，由于yue前面用到了“%”，因此该查询必然走全表扫描，除非必要，否则不要在关键词前加%**

**12、数据类型的隐式转换对查询效率的影响**

**sql server2000的数据库，我们的程序在提交sql语句的时候，没有使用强类型提交这个字段的值，由sql server 2000自动转换数据类型，会导致传入的参数与主键字段类型不一致，这个时候sql server 2000可能就会使用全表扫描。Sql server2005上没有发现这种问题，但是还是应该注意一下。**

**13、SQL Server 表连接的三种方式**

1. **Merge Join**
2. **Nested Loop Join**
3. **Hash Join**

**SQL Server 2000只有一种join方式——Nested Loop Join，如果A结果集较小，那就默认作为外表，A中每条记录都要去B中扫描一遍，实际扫过的行数相当于A结果集行数x B结果集行数。所以如果两个结果集都很大，那Join的结果很糟糕。**

**SQL Server 2005新增了Merge Join，如果A表和B表的连接字段正好是聚集索引所在字段，那么表的顺序已经排好，只要两边拼上去就行了，这种join的开销相当于A表的结果集行数加上B表的结果集行数，一个是加，一个是乘，可见merge join 的效果要比Nested Loop Join好多了。**