链滴

【转】SQL 优化

作者：honglan
原文链接：https://ld246.com/article/1495545763148
来源网站：链滴

<h2 id="问题的提出">问题的提出</h2>
　在应用系统开发初期，由于开发数库数据比较少，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种原则来删除索引，这有助于写出高性能的 SQL 语句。
<h2 id="SQL语句编写注意问题">SQL 语句编写注意问题</h2>
　　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，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 LAT_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，
<pre><code class="highlight-chroma">　select * from employss where first_name||''||last_name ='Beill Cliton';
</code></pre>
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 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。</
>
<pre><code class="highlight-chroma">　*** where first_name ='Beill' and last_name ='Cliton';
</code></pre>
. 带通配符(%)的 like 语句
　　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 clion 的人。可以采用如下的查询 SQL 语句:
<pre><code class="highlight-chroma">　select * from employee where last_name like '%cliton%';
</code></pre>
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:
<pre><code class="highlight-chroma">　select * from employee where last_name like 'c%';
</code></pre>

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 ord r by 子句中使用表达式。

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，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:

<pre><code class="highlight-chroma">... where not (status ='VALID')
</code></pre>
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 句中显式地加入 NOT 词，NOT 仍在运算符中，见下例:
<pre><code class="highlight-chroma">... where status <>'INVALID';
</code></pre>
对这个查询，可以改写为不使用 NOT:
<pre><code class="highlight-chroma">select * from employee where salary<3000 or salary>3000;
</code></pre>
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 racle 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 racle 对 salary 列使用索引，而第一种查询则不能使用索引。
<hr>
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 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 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, 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 SQLPlus , SQLForms 和 Pro*C 中重新设置 ARRAYSIZE 参数, 可以增加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：
<pre><code class="highlight-chroma">DELETE FROM EMP E WHERE E.ROWID > (SELECT MIN(X.ROWID)

FROM EMP X WERE X.EMP_NO = E.EMP_NO);
</code></pre>
（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 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、were、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 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 ，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。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 ，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：
<pre><code class="highlight-chroma">SELECT TAB_NAME FROM TABLES WHERE (TAB_NAME,DB_VER) = (SELECT
TAB_NAME,DB_VR FROM TAB_COLUMNS WHERE VERSION = 604)
</code></pre>
（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 EXISS(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. 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 OT IN ,我们可以把它改写成外连接(Outer Joins)或 NOT EXISTS.
例子：
<pre><code class="highlight-chroma">
（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 * F

OM EMP (基础表) WHERE EMPNO > 0 AND DEPTNO IN(SELECT DEPTNO FROM DEPT WHERE LOC = 'MELB')
</code></pre>
（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好的方法：
<pre><code class="highlight-chroma">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$SQLARA
WHERE EXECUTIONS>0
AND BUFFER_GETS > 0
AND (BUFFER_GETS-DISK_READS)/BUFFER_GETS < 0.8
ORDER BY 4 DESC

</code></pre>
（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 , DELET , 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,那些不必要的索反而会使查询反应时间变慢.。定期的重构索引是有必要的.：
<pre><code class="highlight-chroma">ALTER INDEX REBUILD
</code></pre>
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. 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后,立刻返回结果. 例子：
<pre><code class="highlight-chroma">(低效):
SELECT DISTINCT DEPT_NO,DEPT_NAME FROM DEPT D , EMP E
WHERE D.DEPT_O = E.DEPT_NO
(高效):
SELECT DEPT_NO DEPT_NAME FROM DEPT D WHERE EXISTS (SELECT 'X'
FROM EMP E WERE E.DEPT_NO = D.DEPT_NO);
</code></pre>
（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

（20） 在 Java 代码中尽量少用接符“+”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”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:
<pre><code class="highlight-chroma">低效:
SELECT ... FROM EPT WHERE SAL * 12 > 25000;
高效:
SELECT ... FROM EPT WHERE SAL > 25000/12;
</code></pre>
（23） 用 >=替代 >
<pre><code class="highlight-chroma">高效:
SELECT * FROM MP WHERE DEPTNO >=4
低效:
SELECT * FROM EP WHERE DEPTNO >3
</code></pre>
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:
<pre><code class="highlight-chroma">SELECT LOC_ID , LOC_DESC , REGION
FROM LOCATION
WHERE LOC_ID = 0
UNION
SELECT LOC_ID , OC_DESC , REGION
FROM LOCATION
WHERE REGION = “MELBOURNE”
</code></pre>
低效:
<pre><code class="highlight-chroma">SELECT LOC_ID , LOC_DESC , REGION
FROM LOCATION
WHERE LOC_ID = 0 OR REGION = “MELBOURNE”
</code></pre>
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同的．
低效:
<pre><code class="highlight-chroma">SELECT.... FROM LOCATION WHERE LOC_ID = 10 OR LOC_ID = 20 OR LOC_ID = 30
</code></pre>
高效
<pre><code class="highlight-chroma">SELECT... FROM LOCATION WHERE LOC_IN IN (10,20,30);
</code></pre>
（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). 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比较将使 ORACLE 停用该索引.
低效: (索引失效)
<pre><code class="highlight-chroma">SELECT ... FROM DEPARTMENT WHERE DEPT_CODE IS NOT NULL;
</code></pre>
高效: (索引有效)
<pre><code class="highlight-chroma">SELECT ... FROM DEPARTMENT WHERE DEPT_CODE >=0;
</code></pre>
（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：
<pre><code class="highlight-chroma">SELECT ACCT_NUM, BALANCE_AMT
FROM DEBIT_TRNSACTIONS
WHERE TRAN_DAE = '31-DEC-95'
UNION
SELECT ACCT_NM, BALANCE_AMT
FROM DEBIT_TRASACTIONS
WHERE TRAN_DAE = '31-DEC-95'
</code></pre>
高效:
<pre><code class="highlight-chroma"><span class="highlight

cl">SELECT ACCT_NUM, BALANCE_AMT
FROM DEBIT_TRA
SACTIONS
WHERE TRAN_DA
E = '31-DEC-95'
UNION ALL
SELECT ACCT_N
M, BALANCE_AMT
FROM DEBIT_TRA
SACTIONS
WHERE TRAN_DA
E = '31-DEC-95'
</code></pre>
（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)
<pre><code class="highlight-chroma"><span class="highlight
cl">SELECT DEPT_CODE FROM DEPT ORDER BY DEPT_TYPE
</code></pre>
高效: (使用索引)
<pre><code class="highlight-chroma"><span class="highlight
cl">SELECT DEPT_CODE FROM DEPT WHERE DEPT_TYPE > 0
</code></pre>
（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.
<pre><code class="highlight-chroma"><span class="highlight
cl">SELECT ... FROM EMP WHERE EMPNO = '123'
</code></pre>
实际上,经过 ORACLE 类型转换, 语句转化为:
<pre><code class="highlight-chroma"><span class="highlight
cl">SELECT ... FROM EMP WHERE EMPNO = TO_NUMBER('123')
</code></pre>
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.
<pre><code class="highlight-chroma"><span class="highlight
cl">SELECT ... FROM EMP WHERE EMP_TYPE = 123
</code></pre>
这个语句被 ORACLE 转换为:
<pre><code class="highlight-chroma"><span class="highlight
cl">SELECT ... FROM EMP WHERETO_NUMBER(EMP_TYPE)=123
</code></pre>
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_AEA_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.下面两个查询返回同结果但第二个明显就快了许多.
低效:
<pre><code class="highlight-chroma">SELECT JOB , AVG(SAL)
FROM EMP
GROUP by JOB
HAVING JOB = ‘RESIDENT'
OR JOB = ‘MANGER'
</code></pre>
高效:
<pre><code class="highlight-chroma">SELECT JOB , AVG(SAL)
FROM EMP
WHERE JOB = ‘PESIDENT'
OR JOB = ‘MANGER'
GROUP by JOB
</code></pre>
<hr>
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。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，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

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”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。</p

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。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，这些题统统不存在了。
十二、在程序编码时使用<a href="https://ld246.com/forward?goto=http%3A%2F%2Flib.cs

n.net%2Fbase%2F20" title="Hadoop知识库" target="_blank" rel="nofollow ugc">大数据的数据库
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_IDENTTY，IDENT_CURRENT 和 IDENTITY。如果可能，不要使用 IDENTITY，因为在有触发器的情况下，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，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 NULable，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。