sql优化

# 一、目的

对数据库参数进行优化所获得的性能提升全部加起来只占数据库应用系统性能提升的40%左右，其余60%的系统性能提升全部来自对数据库应用程序的优化。许多优化专家甚至认为对数据库应用程序的优化可以得到80%的系统性能提升。因此可以肯定，通过优化应用程序来对数据库系统进行优化能获得更大的收益。

对应用程序的优化通常可分为两个方面: 源代码的优化和sql语句的优化。由于涉及到对程序逻辑的改变，源代码的优化在时间成本和风险上代价很高(尤其是对正在使用中的系统进行优化) 。另一方面，源代码的优化对数据库系统性能的提升收效有限，因为应用程序对数据库的操作最终要表现为sql语句对数据库的操作。

# 二、直接对sql语句进行优化的原因

1、sql语句是对数据库(数据) 进行操作的惟一途径，应用程序的执行最终要归结为sql语句的执行，sql语句的效率对数据库系统的性能起到了决定性的作用。

2、sql语句消耗了70%～90%的数据库资源。

3、sql语句独立于程序设计逻辑，对sql语句进行优化不会影响程序逻辑，相对于对程序源代码的优化，对sql语句的优化在时间成本和风险上的代价都很低。

4、sql语句可以有不同的写法，不同的写法在性能上的差异可能很大。

5、sql语句易学，难精通。sql语句的性能往往同实际运行系统的数据库结构、记录数量等有关，不存在普遍适用的规律来提升性能。

# 三、优化数据库的思想

1、关键字段建立索引。

2、使用存储过程，它使sql变得更加灵活和高效。

3、备份数据库和清除垃圾数据。

4、sql语句语法的优化。

5、清理删除日志。

# 四、sql语句优化的原则

不要以为只有select语句是查询。实际上，带有任何where条件的 DML(insert、update、delete)语句中都包含查询要求，在后面的文章中，当说到查询时，不一定只是指select语句，也有可能指dml语句中的查询部分。

我们知道，sql语句同其它语言(如c语言)的语句不一样，它是非过程化(non-procedural)的语句，即当你要取数据时，不需要告诉数据库通过何种途径去取数据，如到底是通过索引取数据，还是应该将表中的每行数据都取出来，然后再通过一一比较的方式取数据(即全表扫描)。

为了实现一个查询，内核必须为每个查询定制一个查询策略，或为取出符合条件的数据生成一个执行计划(execution plan)。典型的，对于同一个查询，可能有几个执行计划都符合要求，都能得到符合条件的数据。例如，参与连接的表可以有多种不同的连接方法，这取决于连接条件和优化器采用的连接方法。为了在多个执行计划中选择最优的执行计划，优化器必须使用一些实际的指标来衡量每个执行计划使用的资源(i/0次数、 cpu等)，这些资源也就是我们所说的代价(cost)。如果一个执行计划使用的资源多，我们就说使用执行计划的代价大。以执行计划的代价大小作为衡量标 准，优化器选择代价最小的执行计划作为真正执行该查询的执行计划，并抛弃其它的执行计划。

## 4.1 选择最有效率的表名顺序

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。首先，扫描第一个表（from子句中最后的那个表 ）并对记录进行排序，然后扫描第二个表（from子句中最后第二个表），最后将所有从第二个表中检索出的记录与第一个表中合适记录进行合并。

基础表（driving table)是指被最先访问的表(通常以全表扫描的方式被访问）。

### 4.1.1 当仅有两个表连接查询，选择记录较少的作为基础表。

例：oracle下测试，表tab1有166384条记录，表tab2有1条记录。

1°选择tab2作为基础表 (最好的方法)

select count(\*) from t1,t2 执行时间0.96秒

2°选择tab1作为基础表 (不佳的方法)

select count(\*) from t2,t1 执行时间26.09秒

### 4.1.2 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。

例：t0表描述了t1表和t2表的交集。

高效率：

select \* from t0,t1,t2

where t0.no between 10 and 20 and t0.ci\_no = t2.ci\_no and t0.id =t1.id

低效率：

select \* from t0 ,t1 , t2

where t0.ci\_no = t2.ci\_no and t0.id =t1.id and t0.no between 10 and 20

## 4.2 where子句中的连接顺序

数据库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。同时在链接的表中能过滤的就应该先进行过滤。例如:

(低效,执行时间156.3秒)

**select … from emp e where sal > 50000 and job =‘manager’and 25 < (select count(\*) from emp where mgr=e.empno);**

(高效,执行时间10.6秒)

**select … from emp e where 25 < (select count(\*) from emp where mgr=e.empno) and sal > 50000 and job = ‘manager’;**

## 4.3 select子句中避免使用 ‘ \* ’

当你想在select子句中列出所有的column时,使用动态sql列引用 ‘\*’ 是一个方便的方法.但这是一个非常低效的方法. 实际上,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.

## 4.3 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

## 4.5 计算记录条数

和一般的观点相反, count(\*) 比count(1)稍快 , 当然如果可以通过索引检索,对索引列的计数仍旧是最快的. 例如 count(empno)

## 4.6 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，例如:

低效：

select region，avg(log\_size) from location group by region having region region != ‘sydney’ and region != ‘perth’

高效：

select region，avg(log\_size) from location where region region != ‘sydney’ and region != ‘perth’ group by region；

having 中的条件一般用于对一些集合函数的比较,如count() 等等. 除此而外,一般的条件应该写在where子句中

## 4.7通过内部函数提高sql效率

## 4.8 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。

## 4.9用exists替代in

## 4.10用not exists替代not in

## 4.11 用表连接替换exists

通常来说 , 采用表连接的方式比exists更有效率

select ename from emp e where exists (select‘x’from dept where dept\_no = e.dept\_no and dept\_cat =‘a’);

更高效：select ename from dept d,emp e where e.dept\_no = d.dept\_no and dept\_cat = ‘a’;

## [4.12用索引提高效率](索引.docx)

索引是表的一个概念部分,用来提高检索数据的效率.

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.

## 4.13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（**非常实用的规则，请务必牢记**），举例:

低效：select a from t1 where a \* 5 > 100;

高效：select a from t1 where a >100/5;

## 4.14 自动选择索引

如果表中有两个及以上索引，且有一个是唯一性索引，而其他非唯一。在这种情况下，oracle将使用唯一性索引而完全忽略非唯一性索引。举例:

select ename from emp here empno = 2326 and deptno = 20 ;

这里，只有empno上的索引是唯一性的，所以empno索引将用来检索记录．

table access by rowid on emp index unique scan on emp\_no\_idx

## 4.15 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。

## 4.16 用“>=”替代“>”

如果deptno上有一个索引,

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.

## 4.17 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a和c上都建有索引。

高效：

select a, b, c from t1 where a = 1 union select a, b, c from t1 where c = 2;

低效：

select a, b, c from t1 where a = 1 or c = 2;

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.

## 4.18 用in来替换or

低效：select …. from t1 where a = 10 or a = 20 or a = 30

高效：select … from t1 where a in (10,20,30);

## 4.19 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

## 4.20 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.

## 4.21 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。union将对结果集合排序,这个操作会使用到sort\_area\_size这块内存

union：如果查询出来的结果中有重复记录，那么就去重，关键字"distinct"

union all：就显示所有的符合条件的记录，重复也保留

## 4.22 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.

## 4.23 避免改变索引列的类型.

当比较不同数据类型的数据时,自动对列进行简单的类型转换.

假设 a是一个数值类型的索引列.

select … from t1 where a =”1”

实际上,经过oracle类型转换, 语句转化为:

select … from t1where a = 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b是一个字符类型的索引列

select … from t2 where b = 123

这个语句被oracle转换为:

select … from t1 where to\_number(b)=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会优先转换字符类型到数字类型。

## 4.24 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, ‘!=’ 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**。 ‘+’是数学函数. 就像其他数学函数那样, 停用了索引。

## 4.25 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。