链滴

性能优化（二）数据库优化

作者：Heayan
原文链接：https://ld246.com/article/1520930674758
来源网站：链滴

原文地址 https://www.heayan.com/articles/2018/03/12/1520870418626.html

1. PLSQL程序优化原则

1.1 导致性能问题的内在原因

导致系统性能出现问题从系统底层分析也就是如下几个原因：

- CPU占用率过高，资源争用导致等待
- 内存使用率过高，内存不足需要磁盘虚拟内存
- IO占用率过高，磁盘访问需要等待

1.2 PLSQL优化的核心思想

PLSQL优化实际上就是避免出现“导致性能问题的内在原因”，实际上编写程序，以及性能问题跟踪该本着这个核心思想去考虑和解决问题。

PLSQL程序占用CPU的情况：

- 系统解析SQL语句执行，会消耗CPU的使用
- 运算（计算）会消耗CPU的使用

PLSQL程序占用内存的情况：

- 读写数据都需要访问内存
- 内存不足时，也会使用磁盘

PLSQL程序增大IO的情况：

- 读写数据都需要访问磁盘IO
- 读取的数据越多，IO就越大

大家都知道CPU现在都很高，计算速度非常快；访问内存的速度也很快；但磁盘的访问相对前两个相速度就差的非常大了，因此PLSQL性能优化的重点也就是减少IO的瓶颈，换句话说就是尽量减少IO的问。

性能的优先级CPU>内存>IO，影响性能的因素依次递增。根据上面的分析，PLSQL优化的核心思想：

1. 避免过多复杂的SQL脚本，减少系统的解析过程
2. 避免过多的无用的计算，例如：死循环
3. 避免浪费内存空间没有必要的SQL脚本，导致内存不足
4. 内存中计算和访问速度很快
5. 尽可能的减少磁盘的访问的数据量，该原则是PLSQL优化中重要思想。
6. 尽可能的减少磁盘的访问的次数，该原则是PLSQL优化中重要思想。

下面具体介绍常见影响性能的SQL语句情况。

1.3 Oracle优化器

ORACLE的优化器有RULE (基于规则)、COST (基于成本)、CHOOSE (选择性) 三种。

设置缺省的优化器,可以通过对init.ora文件中OPTIMIZER_MODE参数的各种声明,如RULE,COST,CHOOSE,ALL_ROWS,FIRST_ROWS。你当然也在SQL句级或是会话(session)级对其进行覆盖。

为了使用基于成本的优化器(CBO, Cost-Based Optimizer) , 你必须经常运行analyze 命令,以增加数库中的对象统计信息(object statistics)的准确性。

如果数据库的优化器模式设置为选择性(CHOOSE),那么实际的优化器模式将和是否运行过analyze命有关. 如果table已经被analyze过, 优化器模式将自动成为CBO,反之,数据库将采用RULE形式的优化器。

在缺省情况下,ORACLE采用CHOOSE优化器,为了避免那些不必要的全表扫描(full table scan),你必须量避免使用CHOOSE优化器,而直接采用基于规则或者基于成本的优化器。

在oracle10g前默认的优化模式是CHOOSE，10g默认是ALL_ROWS，我不建议大家去改动ORACLE默认优化模式。

1.4 PLSQL优化

主要说明了在SQL编写上和PLSQL程序编写上可以优化的地方。

1.4.1 选择最有效率的表名顺序

只在基于规则的优化器rule中有效，目前我们oracle选择的优化器基本都不选择rule，因此该问题基不会出现，但为了安全和规范起见，建议编程习惯采用该规则。

ORACLE的解析器按照从右到左的顺序处理FROM子句中的表名,因此FROM子句中写在最后的表(基表 drivingtable)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基础表.当ORACLE处理多个表时, 会运用排序及合并的方式连接它们.首先,扫描第一个表(FROM子句最后的那个表)并对记录进行派序,然后扫描第二个表(FROM子句中最后第二个表),最后将所有从第二表中检索出的记录与第一个表中合适记录进行合并。

```
<html>
例如:<br/>
表 ac01有 16,384 条记录 <br/>
表 ab01 有1 条记录<br/>
<br/>
选择ab01作为基础表 (好的方法)<br/>
select count(*) from ac01,ab01 执行时间0.96秒<br/>
<br/>
选择ac01作为基础表 (不好的方法)<br/>
select count(*) from ab01,ac01执行时间26.09秒
</html>
```

1.4.2 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条之前。

低效:

```
SELECT ab01.aab001,ab02.aab051
FROM ab01,ab02
WHERE ab02.aae140=’31’ AND ab01.aab001=ab02.aab001;
```

高效:

```
SELECT ab01.aab001,ab02.aab051
FROM ab01,ab02
WHERE ab01.aab001=ab02.aab001
AND ab02.aae140=’31’;
```

1.4.3 SELECT子句中避免使用 ‘*’

<html>
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.不幸的是这是一个非常低效的方法. 实际上,ORACLE在解析的过程中, 会将'*' 依次转换成所有的列名, 这个工作通过查询数据字典完成的, 这意味着将耗费更多的时间。
</html>

1.4.4 用EXISTS替代IN

实际情况看，使用exists替换in效果不是很明显，基本一样。在许多基于基础表的查询中,为了满足一条件,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。

低效:

```
SELECT * FROM ac01
Where aac001 in (
   select aac001 from ac02
   where aab001=str_aab001 and aae140=’31’
);
```

或

```
SELECT * FROM ac01
Where aac001 in (
   select distinct aac001 from ac02
   where aab001=str_aab001 and aae140=’31’
);
```

注意使用distinct也会影响速度。

高效:

```
SELECT * FROM ac01
Where exists (
   select 1 from ac02
   where aac001=ac01.aac001 and aab001=str_aab001 and aae140=’31’
```

);

in的常量列表是优化的(例如：aab019 in（‘20’,’ 30’)),不用exists替换；in列表相当于or。

1.4.5 用NOT EXISTS替代NOT IN

Oracle在10g之前版本not in都是最低效的语句，虽然在10g上not in做到了一些改进，但仍然还是存一些问题，因此我们一定要使用not exists来替代not in的写法。

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NOT EXISTS.

低效：

```
SELECT * FROM ac01
WHERE aab001 NOT IN (
   SELECT aab001 from ab01
   where aab020=’ 100’
);
```

高效：

```
SELECT * FROM ac01
WHERE not exists (
   SELECT 1 from ab01 where aab001=ac01.aab001 and aab020=’ 100’
);
```

1.4.6 用表连接替换EXISTS

在子查询的表和主表查询是多对一的情况，一般采用表连接的方式比EXISTS更有效率。

低效：

```
SELECT ac01.* FROM ac01
Where exists (
   select 1 from ac02
   where aac001=ac01.aac001 andaab001=ac01.aab001 and andaae041='200801'
);
```

高效：

```
SELECT ac01.* FROM ac02,ac01
Where ac02.aac001=ac01.aac001 and ac02.aab001=ac01.aab001 and ac02.aae140='31' and a
e041='200801';
```

到底exists和表关联哪种效率高，其实是根据两个表之间的数据量差别大小是有关的，如果差别不大际上速度基本差不多。

1.4.7 用EXISTS替换DISTINCT

当提交一个包含一对多表信息(比如个人基本信息表和个人参保信息表)的查询时,避免在SELECT子句中用DISTINCT.一般可以考虑用EXISTS替换。

低效:

```
select distinct ac01.aac001
from ac02,ac01
where ac02.aac001 = ac01.aac001 and ac02.aae140='31' and ac01.aab001='100100';
```

高效:

```
select ac01.aac001 from ac01
where exists(
   select 1 from ac02 where aac001 = ac01.aac001 and aae140='31'
) and ac01.aab001='100100';
```

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。因此如果是特别研究和追求速度的话（例如：数据转换），查询一个表的数据需要关联其他表的这种情况查询建议采用EXISTS的方式。

1.4.8 减少对表的查询

该问题是我们编程中出现过的问题，请大家一定注意，并且该类问题优化可以带来较大性能的提升。

低效：

```
cursor cur_kc24_mz is
   Select akc260 from kc24
   where akb020 =str_akb020 and aka130=’11’;
cursor cur_kc24_zy is
   Select akc260 from kc24
   where akb020 =str_akb020 and aka130=’21’;
for rec_mz incur_kc24_mz loop
  门诊处理.....
end loop;
for rec_mz in cur_kc24_zy loop
  住院处理.....
end loop;
```

高效：

```
cursor cur_kc24 is
Select akc260,aka130 from kc24
where akb020 =str_akb020 and aka130 in (’11’,’21’);
for rec_kc24 in cur_kc24 loop
   if rec_kc24.aka130=’11’ then
      门诊处理.....
   end if;
   if rec_kc24.aka130=’21’ then
      住院处理.....
```

```
    end if;
end loop;
```

高效的做法使用同样的条件（或者说是索引）只访问一次磁盘，低效的做法访问了2次磁盘，这样速差别将近2倍。

1.4.9 避免循环（游标）里面嵌查询

游标里面不能嵌入查询(或者再嵌游标)，其实也不能有update delete等语句，只能有insert语句。但实际的编程情况下是不可能完全避免的，但我们一定要尽量避免。该类问题也是我们程序中出现过的题，该类问题也可以大大提升程序效率，请大家一定注意。

低效：

```
Cursor cur_ac04 is
  Select aac001,akc010 From ac04
  Where aab001= prm_aab001;
……
For rec_ac04 in cur_ac04 loop
   Select aac008 Into str_aac008 from ac01
    where aac001=rec_ac04.aac001;
if str_aac008=’1’ then
    n_jfje := rec_ac04.akc010*0.08;
end if;
if str_aac008=’2’ then
    n_jfje := rec_ac04.akc010*0.1;
end if;
End loop;
```

高效：

```
Cursor cur_ac04 is
    Select ac01.aac001,ac04.akc010,ac01.aac008 From ac04,ac01
    Where ac04.aac001=ac01.aac001 and aab001=prm_aab001;
……
For rec_ac04 in cur_ac04 loop
    if rec.aac008=’1’ then
        n_jfje := rec_ac04.akc010*0.08;
    end if;
    if rec.aac008=’2’ then
        n_jfje := rec_ac04.akc010*0.1;
    end if;
end loop;
```

优化的方法是尽量把游标循环中的查询语句放到游标查询中一起查询出来，这样相当于只访问了1次盘读到内存；如果放到游标中的话，假如游标有100万数据量，那么程序需要100万次磁盘，可以想浪费了多少IO的访问。

如果在程序编写上没有办法避免游标中有查询语句的话（一般情况是可以避免的），那么也要保证游中的查询使用的索引（即查询速度非常快），例如：游标100万数据量，游标中的查询语句执行需要0.2秒，从这个速度上来说是很快的，但总体上看100万*0.02秒=2万秒=5小时33分钟，如果写一个不优化的语句需要1秒，那么需要几天能执行完呢？

1.4.10 尽量用union all替换union

Union会去掉重复的记录，会有排序的动作，会浪费时间。因此在没有重复记录的情况下或可以允许重复记录的话，要尽量采用union all来关联。

1.4.11 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。

低效：

```
select count(1) from ac01
where aab001=’ 100001’ and aac008=’ 1’ ;

select count(1) from ac01
where aab001=’ 100001’ and aac008=’ 2’ ;
```

低效：

```
Select count(1),aac008 From ac01
Where aab001=’ 100001’ and aac008 in (’ 1’ ,’ 2’ )
group by aac008;
```

高效：

```
select count(decode(aac008,’ 1’ ,’ 1’ ,null)) zz,
count(decode(aac008,’ 2’ ,’ 1’ ,null))tx from ac01
where aab001=’ 100001’ ;
```

特别说明：

group by和order by 都会影响性能，编程时尽量避免没有必要的分组和排序，或者通过其他的有效编程办法去替换，比如上面的处理办法。

1.4.12 group by优化

Group by需要查询后排序，速度慢影响性能，如果查询数据量大，并且分组复杂，这样的查询语句性能上是有问题的。尽量避免使用分组或者采用上面的一节的办法去代替。采用group by的也一定进行优化。

低效：

```
selectac04.aac001,ac01.aac002,ac01.aac003,sum(aac040),ac01.aab001 from ac04,ac01
where ac04.aac001=ac01.aac001 andac01.aab001='1000000370'
group by ac04.aac001,ac01.aac002,ac01.aac003,ac01.aab001;
```

高效：

```
selectac04.aac001,ac01.aac002,ac01.aac003,gzze,ac01.aab001 from (
```

```
    select aac001,sum(aac040) gzzefrom ac04
    group by aac001
) ac04,ac01
where ac04.aac001=ac01.aac001 and aab001='1000000370';
```

1.4.13 尽量避免用order by

Order by需要查询后排序，速度慢影响性能，如果查询数据量大，排序的时间就很长。但我们也不能免不使用，这样大家一定注意一点的是如果使用order by那么排序的列表必须符合索引，这样在速度会得到很大的提升。

1.4.14 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。

低效:

```
SELECT aac008,count(1) FROM ac01
GROUP BY aac008
HAVING aac008 in ( ‘1’ ,’ 2’ );
```

高效:

```
SELECT aac008,count(1) FROM ac01
Where aac008 in( ‘1’ ,’ 2’ )
GROUP BY aac008 ;
```

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HERE子句中

1.4.15 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,就可以减少析的时间并减少那些由Column歧义引起的语法错误.

1.4.16 删除重复记录

一般数据转换的程序经常会使用到该方法。最高效的删除重复记录方法 (因为使用了ROWID)

```
DELETE FROM ac01 a
WHERE a.rowid > (
    SELECT MIN(b.rowid) FROM ac01 b
    WHERE a.aac002=b.aac002 and a.aac003=b.aac003
);
```

1.4.17 COMMIT使用

数据转换的程序需要关注这一点。

1. Commit执行也是有时间的，不过时间特别短，但提交频率特别大，必然也会浪费时间。

2. commit可以释放资源，在大量数据更新时，必须及时提交。

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

例如：

Cur_ac20有5000万数据

```
n_count :=0;
For arec in cur_ac20 loop
   Insertinto ac20 ……
   n_count := n_count + 1;
   If n_count = = 100000 then  --10万一提交
      commit;
      n_count := 0;
   End if;
End loop;
Commit;
```

如果1条一提交，需要提交5000万必然浪费时间；如果整体提交，资源不能释放，性能必须下降。在际编程时，应注意提交的次数和提交的数据量的平衡关系。

1.4.18 减少多表关联

表关联的越多，查询速度就越慢，尽量减少多个表的关联，建议表关联不要超过3个（子查询也属于关联）。

数据转换上会存在大数据量表的关联，关联多了会影响索引的效率，可以采用建立临时表的办法，有更能提高速度。

1.4.19 批量数据插入

数据转换时或者大业务数据插入时，有以下几种办法进行数据插入（不包括imp、impdp和sqlloade）

1、Insert into …select 方式

将查询的结果一次插入到目标表中。

例如：

```
Insert into ac01_bakselect * from ac01;
```

由于是一次查询一次插入，并且最后一次提交，他的速度要比下面描述的curosr的方式速度要快。但询插入的数据量过大必然会占用更多的内存和undo表空间，只能在插入完成后提交，这样资源不能放，会导致回滚表空间不足和快照过旧的问题，另外一旦失败需要全部回滚。因此建议小数据量（例

：300万以下）的导入采用该种方式。

2、Insert /*+append */ into … select方式

该种方式同上种方式，不过由于有append的提示，这种语句不走回滚段直接插入数据文件，速度非快。注意系统开发编程不能使用该种方式，数据转换可以灵活使用。

3、Cursor方式

定义游标，然后逐行进行插入，然后定量提交。

例如：

```
Cusor cur_ac20 is
  Select* from ac20;
....
n_count :=0;
For rec_ac20 in cur_ac20 loop
   Insertinto ac20_bak(aac001,.....) Values(rec_ac20.aac001,....);
   If n_count :==100000 then
      Commit;
      n_count :=0;
   End if;
End loop;
```

4、批绑定的方式

通过游标查询将数据逐行写到数组里（实际上就是内存），然后通过批绑定的语句forall … in… insert nto…values…;将内存的数据一次写入到数据文件中。相比cursor的方式减少了对io的访问次数，提高速度，但注意内存别溢出了。

1.5 索引使用优化

在实际的应用系统中索引问题导致性能问题可能能占到80%，在程序优化上索引问题是需要我们特别注的。本节主要描述什么情况索引会不生效。

1.5.1 避免在索引列上使用函数或运算

这个问题是在我们实际编程中出现过的，请大家一定注意。在索引列上使用函数或运算，查询条件都会使用索引。

例如：

不使用索引

```
Select * from ka02
where aka060=’ 10001000’  and to_char(aae030,’ yyyymm’ )=’ 200801’ ;
```

使用索引

```
Select * from ka02
```

```
where aka060=’ 10001000’ and aae030=to_date(’ 200801’,’ yyyymm’);
```

不使用索引

```
Select * from ka02
where aka060=’ 10001000’ and aae031+1=sysdate;
```

使用索引

```
Select * from ac04
where aac001=’ 10001000’ and aae031=sysdate-1;
```

如果一定要对使用函数的列启用索引, ORACLE新的功能: 基于函数的索引(Function-BasedIndex)

```
CREATE INDEX IDX_KA02_AKA066 ON KA02 (UPPER(AKA066)); /*建立基于函数的索引*/
SELECT * FROM KA02 WHERE UPPER(AKA066) = ‘ASPL’; /*将使用索引*/
```

不是极特殊情况，建议不要使用。

1.5.2 避免改变索引列的类型.

索引列的条件如果类型不匹配，则不能使用索引。

例如:

不使用索引

```
Select * from ac01
where aac001=10001000;
```

使用索引

```
Select * from ac01
where aac001=’ 10001000’;
```

1.5.3 避免在索引列上使用NOT

避免在索引列上使用NOT,NOT不会使查询条件使用索引。对于!=这样的判断也是不能使用索引的，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

低效: (这里,不使用索引)

```
select * From ac02
Where not aab019=’ 10’;
```

高效: (这里,使用了索引)

select * From ac02

Where aab019 in(‘20’,’30’);

1.5.4 用>=替代>

虽然效果不是特别明显，但建议采用这种方式

低效:

```
SELECT *
FROM ab01
WHERE aab019 >  ‘10’
```

高效:

```
SELECT * FROM ab01
WHERE aab019 >=’20’
```

两者的区别在于, 前者DBMS首先定位到aab019=10的记录并且向前扫描到第一个aab019大于10的录,而后者DBMS将直接跳到第一个aab019等于10的记录

###1.5.5 避免在索引列上使用IS NULL和IS NOT NULL

对于索引列使用is null或is not null不会使用上索引. 因为空值不存在于索引列中,所以WHERE子句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 ab01
where aab019 is not null;
```

高效: (索引有效)

```
select * from ab01
where aab019 in(‘10’,’20’,’30’);
```

在实际开发中，对于这类的问题很难避免，如果不是特别影响速度或者要求速度的，可以忽略。

1.5.6 带通配符（%）的like语句

%在常量前面索引就不会使用。

不使用索引

```
Select * from ac01
where aac002 like  ‘%210104’;
Select * from ac01
where aac002 like  ‘%210104%’;
```

使用索引

```
Select * from ac01
where aac002 like  ‘210104%’;
```

1.5.7 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被where子句引用时,优化器才会选择使用该索引。

例如:

Ac02的复合索引：aac001、aae140、aae041

```
Select * from ac02 where aae140=’31’ and aae041=’200801’; --不会使用索引
Select * from ac02 where aac001=’10001000’; --可以使用索引
```

如果不使用索引第一列基本上不会使用索引，使用索引要按照索引的顺序使用，另外使用复合索引的越多，查询的速度就越快

1.5.8 多个平等的索引

当SQL语句的执行路径可以使用分布在多个表上的多个索引时, ORACLE会同时使用多个索引并在运行对它们的记录进行合并, 检索出仅对全部索引有效的记录。

在ORACLE选择执行路径时,唯一性索引的等级高于非唯一性索引。 然而这个规则只有 当WHERE子句索引列和常量比较才有效.如果索引列和其他表的索引类相比较。 这种子句在优化器中的等级是非常的。

如果不同表中两个相同等级的索引将被引用, FROM子句中表的顺序将决定哪个会被率先使用。 FRO子句中最后的表的索引将有最高的优先级。

如果同一表中有两个相同等级的索引被引用，oracle会分析最有效的索引去引用，其他的索引不会使，如果这些相同等级的索引效果差不多，oracle可能会自动合并进行使用。

1.5.9 不明确的索引等级

当ORACLE无法判断索引的等级高低差别,优化器将只使用一个索引,它就是在WHERE子句中被列在最面的。

1.5.10 自动选择索引

如果表中有两个以上（包括两个）索引，其中有一个唯一性索引，而其他是非唯一性。 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。

1.5.11 使用提示(Hints)

对于表的访问,可以使用两种Hints. FULL 和 ROWID

FULL hint 告诉ORACLE使用全表扫描的方式访问指定表.

例如:

```
SELECT /*+ FULL(AC01) */ *
```

```
FROM AC01
WHERE AAC001 =  ‘10001000’ ;
```

如果一个大表没有被设定为缓存(CACHED)表而你希望它的数据在查询结束是仍然停留

在SGA中,你就可以使用CACHE hint 来告诉优化器把数据保留在SGA中. 通常CACHE hint 和 FULL hin 一起使用。

例如:

```
SELECT /*+ FULL(AC01) CACHE(AC01)*/ *  FROM AC01;
```

ROWID hint 告诉ORACLE使用TABLE ACCESSBY ROWID的操作访问表。

采用TABLE ACCESS BY ROWID的方式特别是当访问大表的时候,使用这种方式, 你需要知道ROIWD 值或者使用索引。

索引hint 告诉ORACLE使用基于索引的扫描方式。 你不必说明具体的索引名称。

例如:

```
SELECT /*+index(IDX_AC01_AAC002)*/ aac001 FROM AC01
WHERE  aac002='21011111111111111111';
```

在不使用hint的情况下, 以上的查询应该也会使用索引,然而,如果该索引的重复值过多而你的优化器是CO, 优化器就可能忽略索引。 在这种情况下, 你可以用INDEX hint强制ORACLE使用该索引。

ORACLE hints还包括ALL_ROWS,FIRST_ROWS, RULE,USE_NL, USE_MERGE, USE_HASH 等等。

使用hint , 表示我们对ORACLE优化器缺省的执行路径不满意,需要手工修改。

这是一个很有技巧性的工作。 除非特定的情况，例如：数据转换，其他情况最好不用。

1.5.12 表上存在过旧的分析

我们现在很多项目都存在性能问题，其中有很多种情况都是由于分析过旧导致ORACLE判断索引级别资源成本上出现问题，会导致ORACLE判断错误不使用索引。我个人觉得这是ORACLE不够完善的地。

解决办法：

第一种办法：删除分析，停止oracle10g的自动分析，但不使用分析，oracle访问数据的CPU消耗就大。

第二种办法：重新分析，但过长时间后，索引是否会再次失效，没有验证过。

1.5.13 表上存在并行

表上存在并行，ORACLE判断索引级别和资源成本上出现问题，会导致ORACLE判断错误不使用索引。

这个问题我不知道有什么好的处理办法，从现场实际应用速度角度比较，我还是选择去掉并行，因为使用索引进行全表扫描肯定是不能忍受的。

1.5.14 关于索引建立

索引的使用是肯定会大大提高查询的速度，但索引其实也是一种数据，它也是存放的用户类型的表空下的，索引建立的越多越大，占用的空间也越大，从用户的环境来说这也不是问题，但如果一个表有多过大的查询，必然会影响insert、delete和update索引列的速度，因为这些操作改变了整个表的索顺序，oracle需要进行调整，这样性能就下降了。因此我们一定要合理的建立好有效的索引，编程也符合索引的规则，而不能是索引符合编程的规则。

案例：

某项目数据转换，采用游标循环insert的方式，总共2000万的数据，总共用了4个小时，原因就是目表里面有很多索引。解决方法是先删除索引再执行转换脚本，结果不用1小时就完成了，建立全部的引不到半个小时。

原因就是第一种方式每次insert都改变索引顺序，共执行改变2000万次，而第二种方式整体上执行索顺序就一次。

2. PLSQL程序性能问题分析方法

本章主要介绍怎样找到出现性能问题PLSQL程序或语句的方法。

2.1 性能问题分析

出现性能问题，我先要从整体进行分析，一般总体上会有几种现象：

- 整个系统运行速度都慢
- 在业务不忙的时候，所有模块都慢
- 只有在业务繁忙的时候，所有模块都慢
- 时快时慢
- 个别业务模块运行速度慢
- 在业务不忙的时候，该模块就慢
- 只有在业务繁忙的时候，该模块才慢
- 时快时慢

一般导致系统性能慢的因素：

- 硬件：客户端、服务器CPU、内存和存储设备配置不符合应用系统要求
- 网络：网速低、丢包
- 操作系统参数设置：参数设置不合理
- 受到其他软件干扰：例如：防火墙、病毒
- Weblogic设置：参数设置不合理
- Oracle设置：内存、表空间、redolog、系统参数设置不合理等
- PLSQL程序：plsql不优化、未使用索引、锁表

在不同现象下，可能导致性能问题的因素：

1. 一般来说，如果在不办理业务的情况下，整个系统性能就慢的话，应该和PLSQL程序优化是没有关的。可能的因素为硬件、网络、操作系统、其他软件干扰、ORACLE设置。

2. 只有在业务繁忙的时候，整体系统性能都慢，有可能的因素有硬件、操作系统设置、WEBLOGIC设

、ORACLE设置、PLSQL程序。如果在sqlplus下做查询都慢，那么就和weblogic没有关系。

3. 一般来说，如果在不办理业务的情况下，个别业务模块速度就慢的话，那么基本上就是PLSQL程序优化或未使用索引造成的。

4. 只有在业务繁忙的时候，个别模块慢，有可能的因素有硬件、操作系统设置、WEBLOGIC设置、OACLE设置、PLSQL程序、锁表。

5. 这里我们重点是说明PLSQL优化、索引优化的问题，其他例如：硬件、网络、操作系统和oracle设等因素我们不进行说明。

PLSQL优化、索引不使用的问题产生的影响：

1. 对于某段不优化的程序或语句频繁或者全表扫描一个表时，它访问磁盘的时间和占用的吞吐量是很的，这就导致系统IO长时间处于忙的状态，导致整个系统性能下降。

2. 对于某段不优化的程序或语句频繁或者全表扫描一个表时，其他的业务程序也访问同一个表时，速将大大下降。

3. 如果是更新表操作时间长，还可能会导致锁等待，导致会话堵塞，weblogic端也出现压力问题，致这个系统性能下降。

我们一般根据这些现象、以及一些方法判断，来初步分析产生性能问题的大致原因的范围。不过对于一点，还是比较困难的，因为产生问题的原因是多种的，并且还有一定的关联。下面的章节介绍我们经断定是PLSQL优化、索引不使用的问题，我们通过什么方法来具体定位问题。

2.2 Expain Plan分析索引使用

在PL/SQLDeveloper等工具有一个ExpainPlan分析的功能，这个功能可以帮助我们分析SQL语句是使用了索引、使用哪些索引和使用索引的效果。

1. 选择explain plan的窗口

2. 在上面栏中输入SQL语句，然后点击工具栏上的EXECUTE执行（或按F8），就会在下面显示Optmiergoal优化器的默认方式（也可手工选择），以及下面的解释计划，从解释计划上能看到哪个条件语使用了索引，哪个没有使用；哪个表使用了索引，使用了哪个索引，哪些表是全表扫描的(TABLE ACCSS FULL)

3. 分析内容说明：

- COST：根据优化程序的基于开销的方法所估计出的操作开销值对于使用基于规则方法的语句该列空该列值没有特定的测量单位它只是一个用于比较执行规划开销大小的权重值
- Cardinality：根据基于开销的方法对操作所访问行数的估计值
- Bytes：根据基于开销的方法对操作所访问字节的估计
- 通过设置，我们还能看到更多的信息，例如：CPU使用、时间等等

全表扫描的(TABLEACCESS FULL)肯定是速度慢的，如果是大数据量的表，那么这个语句是绝对影响能的。

另外使用了索引也不一定性能就高，因为索引使用也有效率的情况，下面列出索引常见的使用类型：

1. INDEX UNIQUE SCAN：唯一索引扫描，速度最快

2. INDEX RANGE SCAN：范围索引扫描，使用这个索引时，就需要看COST、Cardinality、Bytes的小了，如果特别大，有时候还可能速度低于全表扫描的速度。

我们在知道语句有问题，或者我们对语句进行优化时，这个工具是非常有用的。

2.3 TOPSQL分析

程序有性能问题的时候，我们是不知道哪些语句存在性能的问题，尤其你不是开发人员。幸好有一些具可以帮助我们找到这些存在性能问题的语句。

Toad工具和ORACLE9I的企业管理控制台工具可以捕获到这些问题语句（oracle10g中em的功能不好）。在TOAD和ORACLE9I工具中可以查询到TOPSQL顶级SQL的内容，通过CPU、IO吞吐量、占时间等信息的排序，我们可以找到最影响系统性能的语句，通过分析我们可以看到这些语句的解释计。

根据解释计划，我们可以进行语句的优化，我们知道语句后，我们通过plsql的搜索功能就知道存在问语句的程序了。

这个办法使用有个前提条件就是这些问题语句在系统上运行过，并且没有间隔过长的时间，因此最好在实际出现性能问题的ORACLE上不断的去监控，才能捕获到最全的问题语句。

1. ORACLE9i企业管理控制台工具的topsql
2. 系统运行中的所有TOPSQL
3. 分析具体的SQL语句
4. Sql分析的解释计划

2.4 针对性语句搜索

TOPSQL分析也只能是找到未使用索引的语句（实际上这一点基本就足够解决性能的问题了），但是于2.4章节中的很多内容，没有办法进行捕获。

我们可以采用针对性语句搜索，来尽量找到一些问题语句进行优化。在PLSQL工具中有一个Find DBO ject的功能，可以进行搜索。

我们可以搜索的关键特征信息：

```
<html>
- NOT IN<br/>
- UNION<br/>
- GROUP BY<br/>
- ORDER BY<br/>
- > 、 < 、 <><br/>
- Like ‘%<br/>
- From tab 、Update tab、delete<br/>
</html>
```

tab，其中tab是数据量特别大的表，我们可以针对性的检查对大表访问的语句（例如：ac20、ab09、c22、kc24等）。

这种办法很有局限性，不过也是一种检查代码的办法。

2.5 后台存储过程跟踪

以上的各种办法，基本都不能检查出类似2.4.8和2.4.9等问题，当我们不能通过整体上去发现问题的

候，我们对有性能问题程序直接进行后台的跟踪，有时候往往是最有效的。

2.6 性能监控

用户的生产运行环境，数据库一般都是UNIX系统的小机。我们会在操作系统级上进行监控和ORACL的session上进行监控。

UNIX下监控性能的命令：

1. topas可以观察CPU、内存、IO的使用，不过该命令不是所有的UNIX都通用
2. vmstat可以观察CPU和内存的使用情况。例如：vmstat 1
3. iostat可以观察IO的使用情况。例如：iostat 1
4. ps aux | head -25查看cpu占用最高的进程
5. 查询内存使用最高的进行命令：top
6. 查看oracle进程命令：ps –ef|greporacle
7. 通过ps找到进程的pid，我们也可以执行下面的语句查看进程具体运行的SQL语句的文本：

```
SELECT /*+ ORDERED */ sql_text FROM v$sqltext a
WHERE (a.hash_value, a.address) IN (
SELECT DECODE(sql_hash_value,0,prev_hash_value,sql_hash_value),DECODE (sql_hash_value,0,
prev_sql_addr,sql_address)
FROM v$sessionb
WHERE b.paddr = (SELECTaddr FROMv$process c WHERE c.spid ='&pid')
)
ORDER BY piece ASC
```

3. 性能测试工具设计思想

3.1 开发工具的目的

现在所有解决程序上的性能问题的办法，大部分都是在系统运行后的解决办法，我们想制作一个工具该工具在开发阶段和测试阶段就可以直接找到存在问题的程序语句。这样可以大大提高程序上的风险也可以提高代码检查和性能测试的工作量。

3.2 总体设计思想

对后台存储过程的程序文件进行遍历扫描，对SQL语句进行检查。

实现的功能：

1. 检查程序中的SQL语句是否使用索引
2. 查询具有特征的关键字的SQL语句。例如：NOT IN
3. 检查游标中嵌入游标或查询的语句
4. 检查列上使用了函数、运算的语句
5. 检查超过3个表以上的关联语句

6. 检查同一个表多次访问（不容易实现）

3.2.1 读取文件

逐行进行读取

3.2.2 捕获SQL语句

遇到SQL关键字开始记录，直到遇到“；”为止，建立分析表做保存。记录包体名、过程名、简略SQ、详细SQL、是否循环中。

简略SQL：

- SQL去掉回车换行
- 截取前100个字符

3.2.3 SQL语句分析

3.2.4 设置关键字