SQL性能优化

#### 模糊匹配

尽量避免在一个**复杂查询**里面使用 like '%parm1%'

百分号会导致相关列的**索引无法使用**,最好不要用

解决方法:

**A)修改前台程序**

把查询条件的供应商名称一栏由原来的文本输入改为**下拉列表**,用户模糊输入供应商名称时,直接在前台就帮忙定位到具体的供应商

这样在调用后台程序时,这列就可以直接用等于来关联了

**B)直接修改后台**

根据输入条件,先查出符合条件的供应商,并把相关记录保存在一个临时表里头,然后再用临时表去做复杂关联

#### **索引问题**

1. 避免对索引字段进行**计算操作**
2. 避免在索引字段上使用**not,<>,!=**
3. 避免在索引列上使用**IS NULL和IS NOT NULL**
4. 避免在索引列上出现**数据类型转换**
5. 避免在索引字段上使用**函数 必须用可以考虑建函数索引**
6. 避免建立索引的列中使用**空值**
7. 常用的多个列上建立**复合索引**, 注意先后顺序

#### **复杂操作**

部分UPDATE、SELECT 语句 写得很复杂(经常**嵌套多级子查询**)可以考虑适当拆成几步,先生成一些**临时数据表**,再进行关联操作

#### **update**

同一个表的修改在一个过程里出现好几十次

可以**整合**在一个UPDATE语句来完成

### **尽量使用UNION ALL**

UNION 并集 **排序 去重**

UNION ALL 并集

UNION 因为会将各查询子集的记录做比较,故比起UNION ALL ,通常**速度都会慢上**许多

如果使用UNION ALL能满足要求的话,**务必使用UNION ALL**

### **在WHERE 语句中,尽量避免对索引字段进行计算操作**

where trunc(create\_date)=trunc(:date1) 索引用不上

优化:

1. where create\_date>=trunc(:date1) and create\_date <trunc(:date1)+1< pre="">
2. where create\_date between trunc(:date1) and trunc(:date1)+1-1/(24\*60\*60)

### **对Where 语句的法则**

1. 避免在WHERE子句中使用in,not in,or 或者having

可以使用 **exist** 和**not exist**代替 in和not in

可以使用表链接代替 exist

Having可以用**where**代替,如果无法代替可以分两步处理

1. 不要以**字符格式**声明数字,要以**数字格式**声明字符值

(日期同样)否则会使索引无效,产生**全表扫描**

### **对Select语句的法则**

**限制使用select \* from table**

### **排序**

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

### **临时表**

慎重使用临时表可以极大的提高系统性能

### **多表连接**

1. **NESTED LOOP**:一般用在连接的表中有**索引**，并且索引选择性较好的时候.

适用于**驱动表的记录集比较小（<10000）**而且inner表需要有有效的访问方法（Index）。需要注意的是：**JOIN的顺序很重要，驱动表的记录集一定要小**，返回结果集的响应时间是最快的

使用**USE\_NL**(table\_name1 table\_name2)可是强制CBO 执行嵌套循环连接。

1. **HASH JOIN :适用于记录集比较大的情况**

用**USE\_HASH**(table\_name1 table\_name2)提示来强制使用散列连接

1. **SORT MERGE JOIN 用在没有索引，并且数据已经排序的情况**

通常情况下**散列连接**的效果都比排序合并连接要好，然而如果行源已经被排过序，在执行排序合并连接时不需要再排序了，这时排序合并连接的性能会优于散列连接

可以使用**USE\_MERGE**(table\_name1 table\_name2)来强制使用排序合并连接.