**1.查询的模糊匹配**

尽量避免在一个复杂查询里面使用 LIKE '%parm1%'—— 红色标识位置的百分号会导致相关列的索引无法使用，最好不要用.

解决办法:

其实只需要对该脚本略做改进，查询速度便会提高近百倍。改进方法如下：

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

**2.索引问题**

在做性能跟踪分析过程中，经常发现有不少后台程序的性能问题是因为缺少合适索引造成的，有些表甚至一个索引都没有。这种情况往往都是因为在设计表时，没去定义索引，而开发初期，由于表记录很少，索引创建与否，可能对性能没啥影响，开发人员因此也未多加重视。然一旦程序发布到生产环境，随着时间的推移，表记录越来越多

这时缺少索引，对性能的影响便会越来越大了。

这个问题需要数据库设计人员和开发人员共同关注

法则：不要在建立的索引的数据列上进行下列操作:

◆避免对索引字段进行计算操作

◆避免在索引字段上使用not，<>，!=

◆避免在索引列上使用IS NULL和IS NOT NULL

◆避免在索引列上出现数据类型转换

◆避免在索引字段上使用函数

◆避免建立索引的列中使用空值。

**3.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

**4.update**

同一个表的修改在一个过程里出现好几十次，如：

|  |
| --- |
| update table1 set col1=... where col2=...; update table1 set col1=... where col2=... ...... |

象这类脚本其实可以很简单就整合在一个UPDATE语句来完成（前些时候在协助xxx项目做性能问题分析时就发现存在这种情况）

**5.在可以使用UNION ALL的语句里，使用了UNION**

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。一般来说，如果使用UNION ALL能满足要求的话，务必使用UNION ALL。还有一种情况大家可能会忽略掉，就是虽然要求几个子集的并集需要过滤掉重复记录，但由于脚本的特殊性，不可能存在重复记录，这时便应该使用UNION ALL，如xx模块的某个查询程序就曾经存在这种情况，见，由于语句的特殊性，在这个脚本中几个子集的记录绝对不可能重复，故可以改用UNION ALL）

**6.在WHERE 语句中，尽量避免对索引字段进行计算操作**

这个常识相信绝大部分开发人员都应该知道，但仍有不少人这么使用，我想其中一个最主要的原因可能是为了编写写简单而损害了性能，那就不可取了

9月份在对XX系统做性能分析时发现，有大量的后台程序存在类似用法，如：

|  |
| --- |
| ...... where trunc(create\_date)=trunc(:date1) |

虽然已对create\_date 字段建了索引，但由于加了TRUNC，使得索引无法用上。此处正确的写法应该是

|  |
| --- |
| where create\_date>=trunc(:date1) and create\_date<trunc(:date1)+1< pre=""></trunc(:date1)+1<> |

或者是

|  |
| --- |
| where create\_date between trunc(:date1) and trunc(:date1)+1-1/(24\*60\*60) |

注意：因between 的范围是个闭区间（greater than or equal to low value and less than or equal to high value.），

故严格意义上应该再减去一个趋于0的小数，这里暂且设置成减去1秒（1/(24\*60\*60)），如果不要求这么精确的话，可以略掉这步。

**7.对Where 语句的法则**

7.1 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。

例子

|  |
| --- |
| SELECT \*  FROM ORDERS WHERE CUSTOMER\_NAME NOT IN  (SELECT CUSTOMER\_NAME FROM CUSTOMER) |

优化

|  |
| --- |
| SELECT \*  FROM ORDERS WHERE CUSTOMER\_NAME not exist  (SELECT CUSTOMER\_NAME FROM CUSTOMER) |

7.2 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。

例子使用：

|  |
| --- |
| SELECT emp.ename, emp.job FROM emp WHERE emp.empno = 7369; 不要使用：SELECT emp.ename, emp.job FROM emp WHERE emp.empno = ‘7369’ |

**8.对Select语句的法则**

在应用程序、包和过程中限制使用select \* from table这种方式。看下面例子

|  |
| --- |
| 使用SELECT empno,ename,category FROM emp WHERE empno = '7369‘ 而不要使用SELECT \* FROM emp WHERE empno = '7369' |

**9. 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

**10.临时表**

慎重使用临时表可以极大的提高系统性能