Performance Optimizing Strategy of PowerBuilder Application

PowerBuilder 应用的性能优化策略

摘要：本文从数据库设计、数据窗口、数据统计等方面对PowerBuilder应用程序设计的性能优化方法进行了详细阐述。

关键词：PowerBuilder　性能优化

金 鹏　（宁波大学商学院　315211）

在使用PowerBuilder开发管理信息系统的过程中，我们发现用户往往对某些功能的时间性和操作方便性提出比较高的要求。这是由业务特点所决定的，这些业务每天的数据录入量比较大，发生频率高，录入时还需查询许多相关信息。特别是那些面向客户的业务，更要必须在较短的时间内完成。因此，对于应用程序的设计，除了要满足基本需求外，必须考虑如何提高应用程序性能，特别是缩短响应时间。笔者通过几个管理信息系统项目的开发实践，总结了一些优化方法。

1 数据库性能的优化

数据库设计是应用程序设计的基础，其性能直接影响应用程序的性能。数据库性能包括存贮空间需求量的大小和查询响应时间的长短两个方面。一般，为了优化数据库性能，需要对数据库中的表进行规范化。规范化的范式可分为第一范式、第二范式、第三范式、BCNF范式、第四范式和第五范式。虽然一个比一个的规范化程度更高，但是满足第三范式的表结构容易维护的且基本满足实际应用的要求。因此，实际应用中一般都按照第三范式的标准进行规范化。规范化的优点是：降低了数据冗余度，从而减少数据库的存储空间；保证数据增加、删除和修改等操作的有效正常进行，维护了数据一致性。但是，规范化也有缺点：由于将一个表拆分为多个表，在查询时需要多表连接，降低了查询速度。

由于规范化有可能导致查询速度慢的缺点，考虑到一些应用需要较快的响应速度，在设计表时应同时考虑对某些表进行反规范化。反规范化可以使用以下几种方法。

（1）分割表。分割表包括水平分割和垂直分割。水平分割是按照行将一个表分割为多个表，这可以提高每个表的查询速度，但查询、更新时要选择不同的表，统计时要汇总多个表，因此应用程序会更复杂。垂直分割是对于一个列很多的表，若某些列的访问频率远远高于其它列，就可以将主键和这些列作为一个表，将主键和其它列作为另外一个表。通过减少列的宽度，增加了每个数据页的行数，一次I/O操作可以扫描更多行，从而提高了访问每一个表的速度。但是由于造成了多表连接，所以应该在同时查询或更新不同分割表中的列的情况比较少的情况下使用。

（2）保留冗余列。当两个或多个表在查询中经常需要连接时，可以在其中一个表上增加若干冗余的列，以避免表之间的连接。由于对冗余列的更新操作必须对多个表同步进行，所以一般在冗余列的数据不经常变动的情况下使用。

（3）增加派生列。派生列是由表中的其它多个列计算所得，增加派生列可以减少统计运算，在数据汇总时可以大大缩短运算时间。

2 数据窗口性能的优化

数据窗口的功能非常强大，在应用程序中使用的很多。因此，数据窗口性能的优化对于提高整个应用程序的性能是非常重要的。优化数据窗口主要是减小查询、更新特别是查询产生的结果集大小和结果集生成的时间，笔者在实践中总结了以下三个方法。

2.1 数据窗口的数据源定义优化

数据窗口的数据源定义，即SQL语句的优化是一个容易被忽视的问题，但却是造成性能问题的一个重要原因。实现同一个查询结果集往往有不同的表达方法，而有些方法的响应时间会比较长。使用这样的表达方法会降低应用程序性能，因此应比较不同表达方法的响应速度。提高响应速度的考虑因素较多，下面列出一些方法。

（1）不要选择不需要的列，包括不显示或者程序中不使用的列，因为检索多个列会花费更多时间。

（2）避免使用不兼容的数据类型。对于不兼容的数据类型，如int和float、double，char和varchar等，SQL语句在执行时需要对数据进行转化，这要占用一些时间。如果在SQL语句中事先把数据转化为相同的数据类型，则可以节省时间。

（3）尽量避免对where子句中的条件参数使用函数或运算符，因为这样优化器将不能使用分步统计信息。例如，SQL语句select name where substring(customerid,1,1)='1' from customer可以优化为select name from customer where customerid like '1%'。

（4）尽量避免使用!和<>等运算符。由于B－树的遍历方式不适用于不等比较，所以这些操作符不能用于索引选择的优化。

（5）当条件参数有多个可能值时，使用IN运算符，不要使用多个OR运算符。例如，SQL语句select name from customer where type='大客户' or type='经销商' 可以优化为select name from customer where type in('大客户','经销商')。

2.2 使用共享数据窗口

当一个窗口中的多个数据窗口检索相同的表，只是查询列不同或数据显示方式不同时，可以使用共享数据窗口，即多个数据窗口使用同一数据缓冲区。其中一个数据窗口作为主数据窗口，负责检索、更新数据，其它数据窗口共享主数据窗口的数据。这样可以避免相同数据检索多次，并简化多数据窗口协调的程序设计。使用共享数据窗口必须保证不同数据窗口的SQL语句定义中，列的名称、排列顺序完全一致。共享数据窗口是通过ShareData函数实现，其返回值等于1表示成功，程序示例如下：

```
int li_flag
li_flag=dw_primary.ShareData(dw_secondly)
if li_flag <> 1 then messagebox("共享错误","共享错误",exclamation!)
```

2.3 动态修改数据窗口

用户经常希望按不同条件查询数据，按不同条件查询的数据窗口在用户界面形式上可能是一样的。对于开发者来讲，会采取两种做法：

（1）使用同一个数据窗口，在检索出所有数据后按条件过滤；

（2）使用不同的数据窗口，即将一个数据窗口另存后修改其数据源的where子句定义。当用户在一个窗口中需要按多种条件查询时，第一种做法是比较合适的，但是若不同的窗口已经限定了查询条件，这种做法会由于检索了大量不需要的数据而降低效率。第二种做法的缺点就是重用性不好，会产生许多很类似的数据窗口，修改时会影响到多个数据窗口。

为了减少类似数据窗口数量，同时不降低查询效率，可以使用modify函数动态修改数据窗口的数据源，主要是where子句部分。其原理是先定义一个最基本的数据窗口，并不带任何查询条件。对于需要查询条件的数据窗口，程序在运行时修改基本数据窗口的数据源。这可以通过一个全局函数来实现，其定义如下：

函数名：f_dw_modiselect(datawindow dw_1,string modi_str)

参数：dw_1，修改的数据窗口控件名；modi_str，修改字符串

返回值：修改成功，返回0；修改失败，返回1

函数体如下：

```
string original_select,mod_string,rc,where_flag='no',where_clause,other_flag=''
int li_pos
//保存原数据窗口数据源
original_select = dw_1.Describe("DataWindow.Table.Select")
//判断原数据窗口是否存在where子句（多表连接时）
if pos(upper(original_select),'WHERE')>0 then  where_flag='yes'
//查找是否包含order_by，having，group_by子句，以确定where子句的插入位置
if pos(upper(original_select),'ORDER BY')>0 then other_flag='ORDER BY'
if pos(upper(original_select),'HAVING')>0 then other_flag='HAVING'
if pos(upper(original_select),'GROUP BY')>0 then other_flag='GROUP BY'
choose case other_flag
 case ''
  if where_flag='yes' then
   where_clause=' and ' + modi_str
  else
   where_clause=' where ' + modi_str
  end if
  mod_string = "DataWindow.Table.Select=' " + original_select + where_clause + " '"
 case else
  if where_flag='yes' then
   //确定where子句插入位置
   li_pos=pos(upper(original_select),other_flag)
   where_clause=mid(original_select,1,li_pos - 1) + ' and ' + modi_str + ' ' + mid(original_select,li_pos)
  else
   where_clause=mid(original_select,1,li_pos - 1) + ' where ' + modi_str + ' ' + mid(original_select,li_pos)
  end if
  mod_string = "DataWindow.Table.Select=' " + where_clause + " '"
```

```
end choose
//修改数据源，rc=''表示修改成功
rc= dw_1.Modify(mod_string)
//省略：根据rc值判断是否修改成功，并返回值
```

3 数据统计的优化

某些数据统计由于需要同时检索多个表并进行大量计算，往往需要很长时间，对于这类统计应考虑进行优化。统计生成的报表按生成频率可分为实时统计报表和月报、年报等报表，实时统计报表的统计生成数据随时可能变化，月报、年报等报表的统计生成数据则不变。由于特点不同，对这两类统计的优化方法也略有不同。

数据统计的优化是按照数据仓库的思想，把统计产生的数据保存在专门的表中。对于实时统计报表，该表中应有一个标志字段表示数据是否是最新的。优化算法是：

（1）首先判断该表是否存在数据，若不存在，进行计算，否则，转到（2）；

（2）判断数据是否是最新的，若不是，删除原数据，重新计算，否则，转到（3）；

（3）若数据是最新的，直接从表中检索。

对于月报、年报等报表，可选择空闲时间计算统计数据，用户在使用时直接检索即可。一旦这些统计数据被保存，查询历史数据和分析、预测就非常方便。

4 结语

应用程序性能的优化包括很多方面，以上所阐述的只是一部分方法，这些方法在开发实践中对程序性能的改善很明显。为了全面提高应用程序性能，还应考虑程序设计算法的优劣、数据库服务器性能、客户端配置等等。

参考文献

1 （美）德卢克（Deluca，S.A）等著，蒋蕊等译；Microsoft SQL Server 7性能优化，机械工业出版社，2000.8。

2 Sharon Bjeletich等著，熊桂喜等译，Microsoft SQL Server 7.0 开发指南，清华大学出版社，2000.9。

3 桂峰等，PowerBuilder 7应用与开发，机械工业出版社，1999.11。

4 刘增进，PowerBuilder 7.0 数据窗口技术详解，电子工业出版社，2000.3。