基于 SQL 语句优化数据查询

Data Inquiry of Optimization on the Basis of SQL Sentence

周 斌 （舟山 浙江海洋学院教务处 316000）

摘要：在数据库系统中，数据查询是一极其重要的操作。影响数据库系统性能的因素有很多，其中运用 SQL 语句的优劣对数据库系统的性能有直接的影响。本文介绍了编写 SQL 语句和提高数据库查询速度需要注重的问题。详细论述了笔者提出的几种数据库优化查询技术，并进行了分析与总结。

关键词：数据库 SQL 语句 优化查询

1 背景知识

数据库系统是管理信息系统的核心，基于数据库的联机事务处理（OLTP）以及联机分析处理（OLAP）是银行、企业、政府等部门最为重要的计算机应用之一。从大多数系统的应用实例来看，查询操作在各种数据库操作中所占据的比重最大，而查询操作所基于的 SELECT 语句在 SQL 语句中又是代价最大的语句。举例来说，如果数据的量积累到一定的程度，比如一个银行的账户数据库表信息积累到上百万甚至上千万条记录，全表扫描一次往往需要数十分钟，甚至数小时。如果采用比全表扫描更好的查询策略，往往可以使查询时间降为几分钟，由此可见查询优化技术的重要性。笔者在应用项目的实施中发现，许多程序员在利用一些前端数据库开发工具（如 PowerBuilder、Delphi 等）开发数据库应用程序时，只注重用户界面的华丽，并不重视查询语句的效率问题，导致所开发出来的应用系统效率低下，资源浪费严重。因此，如何设计高效合理的查询语句就显得非常重要。本文以应用实例为基础，结合数据库理论，介绍查询优化技术在现实系统中的运用。

2 分析问题

许多程序员认为查询优化是 DBMS（数据库管理系统）的任务，与程序员所编写的 SQL 语句关系不大，这是错误的。一个好的查询计划往往可以使程序性能提高数十倍。查询计划是用户所提交的 SQL 语句的集合，查询规划是经过优化处理之后所产生的语句集合。DBMS 处理查询计划的过程是这样的：在做完查询语句的词法、语法检查之后，将语句提交给 DBMS 的查询优化器，优化器做完代数优化和存取路径的优化之后，由预编译模块对语句进行处理并生成查询规划，然后在合适的时间提交给系统处理执行，最后将执行结果返回给用户。在实际的数据库产品（如 Oracle、Sybase 等）的高版本中都是采用基于代价的优化方法，这种优化能根据从系统字典表所得到的信息来估计不同的查询规划的代价，然后选择一个较优的规划。虽然现在的数据库产品在查询优化方面已经做得越来越好，但由用户提交的 SQL 语句是系统优化的基础，很难设想一个原本糟糕的查询计划经过系统的优化之后会变得高效，因此用户所写语句的优劣至关重要。系统所做查询优化我们暂不讨论，下面重点说明改善用户查询计划的解决方案。

3 中解决问题

下面以关系数据库系统 Informix 为例，介绍改善用户查询计划的方法。

3.1 合理使用索引

索引是数据库中重要的数据结构，它的根本目的就是为了提高查询效率。现在大多数的数据库产品都采用 IBM 最先提出的 ISAM 索引结构。索引的使用要恰到好处，其使用原则如下：

（1）在经常进行连接，但是没有指定为外键的列上建立索引，而不经常连接的字段则由优化器自动生成索引。

(2) 在频繁进行排序或分组(即进行 group by 或 order by 操作)的列上建立索引。

(3) 在条件表达式中经常用到的不同值较多的列上建立检索,在不同值少的列上不要建立索引。比如在雇员表的"性别"列上只有"男"与"女"两个不同值,因此就无必要建立索引。如果建立索引不但不会提高查询效率,反而会严重降低更新速度。

(4) 如果待排序的列有多个,可以在这些列上建立复合索引(compound index)。

(5) 使用系统工具。如 Informix 数据库有一个 tbcheck 工具,可以在可疑的索引上进行检查。在一些数据库服务器上,索引可能失效或者因为频繁操作而使得读取效率降低,如果一个使用索引的查询不明不白地慢下来,可以试着用 tbcheck 工具检查索引的完整性,必要时进行修复。另外,当数据库表更新大量数据后,删除并重建索引可以提高查询速度。

3.2 避免或简化排序

应当简化或避免对大型表进行重复的排序。当能够利用索引自动以适当的次序产生输出时,优化器就避免了排序的步骤。以下是一些影响因素:

(1) 索引中不包括一个或几个待排序的列;

(2) group by 或 order by 子句中列的次序与索引的次序不一样。

(3) 排序的列来自不同的表。

为了避免不必要的排序,就要正确地增建索引,合理地合并数据库表(尽管有时可能影响表的规范化,但相对于效率的提高是值得的)。如果排序不可避免,那么应当试图简化它,如缩小排序的列的范围等。

3.3 消除对大型表行数据的顺序存取

在嵌套查询中,对表的顺序存取对查询效率可能产生致命的影响。比如采用顺序存取策略,一个嵌套 3 层的查询,如果每层都查询 1000 行,那么这个查询就要查询 10 亿行数据。避免这种情况的主要方法就是对连接的列进行索引。例如,两个表:学生表(学号、姓名、年龄……)和选课表(学号、课程号、成绩)。如果两个表要做连接,就要在"学号"这个连接字段上建立索引。

还可以使用并集来避免顺序存取。尽管在所有的检查列上都有索引,但某些形式的 where 子句强迫优化器使用顺序存取。下面的查询将强迫对 orders 表执行顺序操作:

```
SELECT * FROM orders WHERE (customer_num = 104 AND order_num > 1001) OR order_num = 1008
```

虽然在 customer_num 和 order_num 上建有索引,但是在上面的语句中优化器还是使用顺序存取路径扫描整个表。因为这个语句要检索的是分离的行的集合,所以应该改为如下语句:

```
SELECT * FROM orders WHERE customer_num = 104 AND order_num > 1001
UNION
SELECT * FROM orders WHERE order_num = 1008
```

这样就能利用索引路径处理查询。

3.4 避免相关子查询

一个列的标签同时在主查询和 where 子句中的查询中出现,那么很可能当主查询中的列值改变之后,子查询必须重新查询一次。查询嵌套层次越多,效率越低,因此应当尽量避免子查询。如果子查询不可避免,那么要在子查询中过滤掉尽可能多的行。

3.5 避免困难的正规表达式

MATCHES 和 LIKE 关键字支持通配符匹配,技术上叫正规表达式。但这种匹配特别耗费时间。例如:SELECT * FROM customer WHERE zipcode LIKE "98_ _ _" 即使在 zipcode 字段上建立了索引,在这种情况下也还是采用顺序扫描的方式。如果把语句改为 SELECT * FROM customer WHERE zipcode > "98000",在执行查询时就会利用索引来查询,显然会大大提高速度。另外,还要避免非开始的子串。例如语句:SELECT * FROM customer WHERE zipcode[2,3] > "80",在 where 子句中采用了非开始子串,因而这个语句也不会使用索引。

3.6 使用临时表加速查询

把表的一个子集进行排序并创建临时表,有时能加速查询。它有助于避免多重排序操作,而且在其他方面还能简化优化器的工作。例如:

```
SELECT cust.name,rcvbles.balance,……other columns
FROM cust,rcvbles
WHERE cust.customer_id = rcvlbes.customer_id
AND rcvblls.balance > 0
AND cust.postcode > "98000"
```

```
ORDER BY cust. name
```

如果这个查询要被执行多次而不止一次，可以把所有未付款的客户找出来放在一个临时文件中，并按客户的名字进行排序：

```
SELECT cust. name, rcvbles. balance, ……other columns
FROM cust, rcvbles
WHERE cust. customer_id = rcvlbes. customer_id
AND rcvblls. balance > 0
ORDER BY cust. name
INTO TEMP cust_with_balance
```

然后以下面的方式在临时表中查询：

```
SELECT * FROM cust_with_balance WHERE postcode > "98000"
```

临时表中的行要比主表中的行少，而且物理顺序就是所要求的顺序，减少了磁盘 I/O，所以查询工作量可以得到大幅减少。

注意：临时表创建后不会反映主表的修改。在主表中数据频繁修改的情况下，注意不要丢失数据。

3.7 用排序来取代非顺序存取

非顺序磁盘存取是最慢的操作，表现在磁盘存取臂的来回移动。SQL 语句隐藏了这一情况，使得我们在写应用程序时很容易写出要求存取大量非顺序页的查询。

有些时候，用数据库的排序能力来替代非顺序的存取能改进查询。

4 实例分析

下面我们举一个制造公司的例子来说明如何进行查询优化。制造公司数据库中包括 3 个表，模式如下所示：

(1) part 表

零件号 (part_num)	零件描述 (part_desc)	其他列 (other column)
10032	Seageat 30G disk	……
500049	Novell10Mnetwork card	……

(2) vendor 表

厂商号 (vendor _num)	厂商名 (vendor_name)	其他列 (other column)
910257	Seageat Corp	
523045	IBMCorp	……

(3) parven 表

零件号 (part_num)	厂商号 (vendor_num)	零件数量 (part_amount)
102032	910257	3450000
500049	523045	4000000

下面的查询将在这些表上定期运行，并产生关于所有零件数量的报表：

```
SELECT part _ desc, vendor _ name, part _ amount
FROM part, vendor, parven
WHERE part. part_num = parven. part_num
AND parven. vendor_num = vendor. vendor_num
ORDER BY part. part_num
```

如果不建立索引，上述查询代码的开销将十分巨大。为此，我们在零件号和厂商号上建立索引。索引的建立避免了在嵌套中反复扫描。关于表与索引的统计信息如下：

表 table	行尺寸 row size	行数量 Row count	每页行数量 Rows/Pages	页数量 Data Pages
part	150	10000	25	400
Vendor	150	1000	25	40
Parven	13	15000	300	50

索引 (Indexes)	键尺寸 (Key Size)	每页键数量 (Keys/Page)	页面数量 (Leaf Pages)
part	4	500	20
Vendor	4	500	2
Parven	8	250	60

看起来是个相对简单的 3 表连接，但是其查询开销是很大的。通过查看系统表可以看到，在 part_num 上和 vendor_num 上有簇索引，因此索引是按照物理顺序存放的。parven 表没有特定的存放次序。这些表的大小说明从缓冲页中非顺序存取的成功率很小。此语句的优化查询规划是：首先从 part 中顺序读取 400 页，然后再对 parven 表非顺序存取 1 万次，每次 2 页（一个索引页、一个数据页），总计 2 万个磁盘页，最后对 vendor 表非顺序存取 1.5 万次，合 3 万个磁盘页。可以看出在这个索引好的连接上花费的磁盘存取为 5.04 万次。实际上，我们可以通过使用临时表分 3 个步骤来提高查询效率：

(1) 从 parven 表中按 vendor_num 的次序读数据：

```
SELECT part_num, vendor_num, price
FROM parven
ORDER BY vendor_num
```

INTO temp pv_by_vn

这个语句顺序读 parven(50 页),写一个临时表(50 页),并排序。假定排序的开销为 200 页,总共是 300 页。

(2) 把临时表和 vendor 表连接,把结果输出到一个临时表,并按 part_num 排序:

```
SELECT pv_by_vn, * vendor. vendor_num
FROM pv_by_vn,vendor
WHERE pv_by_vn. vendor_num = vendor. vendor_num
ORDER BY pv_by_vn. part_num
INTO TMP pvvn_by_pn
DROP TABLE pv_by_vn
```

这个查询读取 pv_by_vn(50 页),它通过索引存取 vendor 表 1.5 万次,但由于按 vendor_num 次序排列,实际上只是通过索引顺序地读 vendor 表(40 +2 =42 页),输出的表每页约 95 行,共 160 页。写并存取这些页引发 5 * 160 =800 次的读写,索引共读写 892 页。

(3) 把输出和 part 连接得到最后的结果:

```
SELECT pvvn_by_pn. * ,part. part_desc
FROM pvvn_by_pn,part
WHERE pvvn_by_pn. part_num = part. part_num
DROP TABLE pvvn_by_pn
```

这样,查询顺序地读 pvvn_by_pn(160 页),通过索引读 part 表 1.5 万次,由于建有索引,所以实际上进行 1772 次磁盘读写,优化比例为 30: 1。笔者在 Informix Dynamic Sever 上做同样的实验,发现在时间耗费上的优化比例为 5: 1(如果增加数据量,比例可能会更大)。

5 小结

20% 的代码用去了 80% 的时间,这是程序设计中的一个著名定律,在数据库应用程序中也同样如此。我们的优化要抓住关键问题,对于数据库应用程序来说,重点在于 SQL 的执行效率。查询优化的重点环节是使得数据库服务器少从磁盘中读数据以及顺序读页而不是非顺序读页。

参考文献

1 唐连章,数据库应用系统的运行效率探讨,广州大学学报(自然科学版),2002。

2 陈继华、王建东、周冬平,基于模拟退火的查询优化算法的设计与实现,计算机应用,2002。

3 李昶、余立人,数据库应用系统性能与数据查询优化,现代计算机,2002。