第16卷 第4期
2006年4月

计算机技术与发展
COMPUTER TECHNOLOGY AND DEVELOPMENT

Vol. 16 No. 4
Apr. 2006

Oracle9i 中查询优化技术的分析

余俊新,孙 涌
(苏州大学 计算机科学与技术学院,江苏 苏州 215006)

摘 要:在一个企业级信息系统中,数据库端的性能对整个系统的有效运行起着关键的作用,而在数据库的性能上,数据查询的效率占据着十分重要的地位。文中根据 Oracle9i 中的数据查询机制以及优化器的特性,探讨了 Oracle9i 中数据库查询语句的优化,并着重对 Oracle9i 中的几种表连接技术进行了分析比较,给出了在实际开发中如何编写高效率的 SQL 查询语句。通过文中的分析结果,实现了应用系统在面临大量的数据时,能够快速对数据进行查询,保障系统的可靠运行。

关键词:优化;优化器;查询优化;性能;表连接
中图分类号:TP311.5 **文献标识码**:A **文章编号**:1005-3751(2006)04-0093-03

Summary of Optimization of Query Statement in Oracle9i

YU Jun-xin, SUN Yong
(School of Computer Science & Technology, Soochow University, Suzhou 215006, China)

Abstract: In an enterprise-class information system, the performance of the database affects the whole system greatly, and on the database side the efficiency of data query affects its performance greatly. According to the mechanism of data query and the trait of the optimizer in Oracle9i, this paper introduces the optimization of data query in Oracle9i, analyzes and compares the different kinds of table connection in Oracle9i emphatically, and presents how to write efficient SQL query statement in the software development. Based on the result of the analysis from this paper, it enables the rapid querying of the data and the steady running of the system when the application system is confronted with a great deal of data.

Key words: optimization; optimizer; query optimization; performance; table connection

0 引 言

在与数据库有关的开发过程中,对于同样一个查询目的,经常可以编写出多种不同的 SQL 语句来满足需要,但是不同的 SQL 语句其执行过程并不同,最终在执行效率上也会存在很大差别。不良的 SQL 语句以低效方式访问数据库,导致不必要的数据扫描和通过网络传输不必要的数据,消耗大量数据库资源。数据库运行中,有相当大的一部分性能问题是由于存在着不良的查询语句使得系统响应变的缓慢。因此,在开发中,提高编写的 SQL 查询语句的质量,能够提高系统性能,保障系统正常运行。

1 Oracle9i 中对 SQL 优化的支持

1.1 Oracle9i 优化器

Oracle 优化器[1]为 SQL 语句选择最佳的执行计划。要对 SQL 语句进行优化,必须了解 Oracle 优化器的工作原理。Oracle 优化器工作原理如下:

(1)分析并估计 SQL 语句的执行时间;

(2)如果 SQL 语句很复杂,优化器可能会修改 SQL 语句使之具有更高效率;

(3)如果 SQL 语句需要访问一个视图,那么某些情况下,在执行优化操作之前,Oracle 优化器将把 SQL 语句中的查询和视图查询结合在一起;

(4)在基于成本的优化方法和基于规则的优化方法之间做出选择;

(5)优化器为 SQL 语句使用的表选择一个或多个访问路径;

(6)优化器选择表的连接顺序;

(7)优化器选择最佳连接操作。

1.2 Oracle9i 中的数据访问方式

Oracle9i 访问数据主要有两种方法[2]:索引和全表扫描。当 Oracle9i 使用全表扫描时,Oracle9i 直接读表中所有的记录来查找所需的记录;当 Oracle9i 使用索引访问数据时,它根据索引值在索引中找到与该值对应的行的 ROWID,然后根据 ROWID 直接找到所需的行。

2 Oracle9i 中的 SQL 查询优化

2.1 合理使用索引

在 Oracle9i 中,索引是用来优化查询的重要技术。当表中有 2%~7% 的数据被检索到的时候,使用索引可以加快查询语句检索数据的速度[3]。但是使用索引也有可

收稿日期:2005-07-25
作者简介:余俊新(1981-),男,湖北荆州人,硕士研究生,研究方向为分布式计算;孙 涌,副教授,研究方向为软件工程。

能带来弊端,比如增加系统时间和空间的开支。因此,对于索引的使用,要注意以下原则:

1)在查询只选择一个表中少量数据时,可以对该表建立索引;

2)不要对经常更新的字段建立索引;

3)具有惟一值的字段最适合建立索引,而对于惟一值很少的字段则不适合建立索引;

4)经常被用来连接表的字段适合建立索引;

5)对于经常需要进行查询的字段建立索引;

6)在大批量的进行记录的插入和删除、更新时,应删除索引;待操作完成后,重新建立索引。

2.2 使用复合索引

如果设计合理,复合索引是非常有用的。在编写SQL查询时,如果注意利用了复合索引,那么对查询性能会产生显著影响。复合索引的编写要注意以下原则:

1)当两个字段本身都不具有惟一性,但两个字段的组合却具有惟一性时,那么为这两个字段创建复合索引比较合适;

2)如果某个 SELECT 语句选择的所有字段都包含在复合索引中,那么在执行查询时,Oracle9i 将不会查询表,而直接从索引中返回数据;

3)建立复合索引时,应将惟一值的较多的字段放在前面,而惟一值较少的放在后面;当多个字段惟一值数目大致相同时,应将经常查询的字段放在前面。

2.3 避免在索引列上使用会禁止索引的关键字

在 Oracle9i 中,如果 SQL 语句包含某些关键字的话,那么即使在该列上加了索引,在查询时,索引也不会被用到。因此,在编写 SQL 语句时,应尽量避免使用这些关键字。表 1 描述了这些关键字与索引的关系。

表 1 关键字与索引的关系图

使用索引的关键字	限制使用索引的关键字
Column between x and y	Column not between a and b
Column in(...)	Column is null
Column like	Column is not null
Column <, >, <=, >=, =	Column not in(…)
Column = x or column = y	Column not like
Column = x and column = y	Comumn = fucktionname()
	Column ! =

2.4 在 ORDER BY,WHERE 子句中不要有非索引项或者计算表达式

Oracle9i 利用 ORDER BY 子句指定的列来对查询结果进行排序,因此在 ORDER BY 子句中有非索引列或者表达式会降低查询速度。在 WHERE 子句中,也有同样的要求,例如,下列 SQL 条件语句中建有适当的索引:SELECT * FROM shijian WHERE time/5 = 1.5。

由于 WHERE 子句中对列的操作是在 SQL 语句运行时逐行得到的,因此必须进行全表扫描。如果这些结果可以在查询编译时得到,那么就可以被 Oracle9i 优化器优化而得以使用索引,可以将上述语句改写为:

SELECT * FROM shijian WHERE time = 7.5

2.5 避免对数据量很大的表进行顺序存取

在查询操作中,顺序存取的效率是很低的,特别是在嵌套查询中。比如,对于一个嵌套三层的查询,如果每一层要查询 100 行数据,那么这个查询将要查询 100 万行数据。避免这种情况的方法是对连接的列建立索引,也可以利用并集来避免顺序存取,因为有时候尽管在连接的列上建立了索引,但是某些 WHERE 子句强迫优化器进行顺序存取。如:

SELECT * FROM orders WHERE customerID = 100 AND orderID > 1001 OR orderID = 1110

对于这条语句,虽然在列 customerID 和 OrderID 上建立了索引,但是优化器还是会使用顺序存取路径扫描整个表,因为这条语句要检索的是分离的行的集合,可以将其进行如下修改来避免这种操作,如:

SELECT * FROM orders WHERE customerID = 100 AND orderID > 1001

UNION

SELECT * FROM orders WHERE orderID = 1110

2.6 避免使用相关子查询

若子查询要引用外层查询的值,则构成相关子查询,执行时就不能象无关子查询那样,先独立处理内层查询,再处理外层查询,而需要外层查询和内层查询交叉进行。先用外层查询得到一行记录,然后执行子查询,得到结果后判断外层查询得到的那条记录是否满足条件,然后再执行外层查询得到下一条记录,如此反复,直到外层查询完毕。这样的查询效率低下,因此应该予以避免。如果子查询不可避免,那么应当在子查询中过滤掉尽可能多的行。例如将

SELECT * FROM A WHERE ID = (SELECT ID FROM A, B WHERE A. ID = B. ID AND B. number > 900)

改写为:

SELECT * FROM A, B WHERE A. ID = B. ID AND B. number > 900

2.7 避免通配符(%)在搜寻词首出现

通配符(%)出现在搜寻词首,系统将不使用索引,会降低 SQL 语句的执行速度。然而当通配符出现在字符串其他位置时,优化器就能利用索引。在下面的查询中,索引得到了使用:

SELECT name FROM authors WHERE state LIKE 'M%'

2.8 强制表扫描

当没有索引或者索引错误,或者表中有要被访问的数据超过 20%时,全表扫描比索引扫面更有效率[3],但是不能直接指定 Oracle 是否使用索引,因此必须编写强制 Oracle 使用表扫描的语句。如果要制止 Oracle 使用索引,可以在 WHERE 子句中做一个假的算术表达式,通常的做法是在数字列上加一个零或者在字符列上加一个 NULL 串。

例如:SELECT * FROM person WHERE age = 24;

改为

SELECT * FROM person WHERE age = 24 + 0;

除了上面介绍的以外,还有其它的一些地方也是需要注意的,如使用临时表加速查询,避免不必要的数据类型转换,避免使用正则表达式,为优化器提供冗余的搜索参数,子查询中尽量使用 EXISTS 而不是 IN,使用 WHERE 代替 HAVING 等等,都可以加快查询的速度。

3 表连接的优化

这一部分专门讨论表连接对查询的影响。表连接从两张或者多张表而不是一张表中收集信息,这些表至少有一列是相同的,通过该列连接。许多 SQL 语句之所以性能低下,主要就是不清楚 Oracle 优化器是如何处理连接两张或者多个表的 SQL 语句的。Oracle9i 中主要有 3 种连接类型:嵌套循环连接,排序合并连接,hash 连接。

3.1 嵌套循环连接

在嵌套循环连接中,Oracle9i 首先从第一张表(驱动表)读取记录,然后把结果集与第二张表进行比较连接。这一切都是通过索引来进行的,如果没有索引,连接成本会大大增加。

3.2 排序合并连接

在排序合并连接中不使用索引,它使用全表扫描来获得两张表的数据集合,并将它们分别排序,合并成为一个最终的结果集。当连接表中有大量数据返回时,排序合并连接比嵌套循环连接要好。

3.3 hash 连接

在 hash 连接中,Oracle9i 访问一张表(通常是连接结果中较小的表),并在内存中建立一张基于连接键的 hash 表,然后扫描连接中的其它表(通常是较大的表),并根据 hash 表检测是否有匹配的记录。由于 hash 算法的存在,是否存在索引对性能的影响不是很大。

3.4 对连接进行优化的途径

3.4.1 选择合适的连接

当为一个包含连接的查询选择执行计划时,Oracle9i 的优化器考虑所有可能的连接方法和查询顺序,估算每一个方案的代价并从中选择一个最佳方案,但有时因为索引的原因,优化器并不能选择一个最优的方案。在这种情况下,可以选择 USE_NL,USE_MERGE,USE_HASH 提示来指定一种连接方法[4]。表 2 对各种连接类型进行了比较,在编写 SQL 查询语句时,可以以此为依据来决定该使用何种连接。

3.4.2 尽早缩减结果集

在连接过程中尽早减少结果集也是加快连接的一个重要方法,也就是尽早使用约束性强的条件,目的是最大程度减少返回的行。Oracle9i 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 条件之前,那些可以过滤掉最大数量记录的条件必须写在 WHERE 子句的末尾。例如,有如下两条 SQL 语句:

表 2　三种表连接的对比

连接类型	嵌套循环连接	排序合并连接	散列连接
优化器提示	USE_NL	USE_MERGE	USE_HASH
使用场合	任何连接	仅用于等价连接	仅用于等价连接
所需资源	CPU,磁盘 I/O	内存,临时表空间	内存,临时表空间
特点	当连接列上存在索引或者搜索条件严格返回的结果集较小时,效率比较高,能够快速返回第一批搜索记录	连接中使用全表扫描,当缺乏索引或者搜索条件模糊,返回结果集较大时,该类型连接比嵌套循环连接有效	连接中使用全表扫描,当缺乏索引或者搜索条件模糊时,该类型连接比嵌套循环连接有效。通常比排序合并连接有效
缺点	当在连接字段上没有建立索引,以及查询条件限制不严返回的结果集很大时,效率很低	所有的表都需要排序,它为最优化的吞吐量设计,在结果没有找到之前不返回数据	建立 hash 表需要大量内存,第一次的结果返回比较慢,如果在磁盘上操作速度将非常慢

```
SELECT … FROM emp E
WHERE 25 < (SELECT COUNT( * ) FROM emp WHERE mgr
= E.empID) AND job = 'MANAGER';
```

将比下列 SQL 语句更有效率:

```
SELECT … FROM emp E
WHERE job = 'MANAGER' AND 25 < (SELECT COUNT( * )
FROM emp WHERE mgr = E.empID);
```

3.4.3 选择合适的表名顺序(只有在基于规则的优化器中才有效)

Oracle9i 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(驱动表)将被最先处理。在 FROM 子句中包含多个表的情况下,必须选择记录数最少的表作为驱动表。当 Oracle9i 处理多个表时,会运用排序合并的方式连接它们。首先,扫描第一个表(FROM 子句中最后的那个表)并对记录进行排序,然后扫描第二个表(FROM 子句中最后第二个表),最后将所有从第二个表中检索出的记录与第一个表中合适记录进行合并。例如,存在两个表,TAB1 包含 10000 条记录,TAB2 包含 100 条记录,以及如下两条 SQL 语句:

```
SELECT COUNT( * ) FROM tab1,tab2
SELECT COUNT( * ) FROM tab2,tab1
```

其中第一条语句的执行效率要比第二条高许多。

如果有 3 个以上的表连接查询,那就需要选择交叉表作为基础表,交叉表是指那个被其他表所引用的表。例如,emp 表描述了 location 表和 category 表的交集:

```
SELECT * FROM location L,category C,emp E
WHERE E.empID BETWEEN 1000 AND 2000 AND E.catID =
C.catID AND E.locn = L.locn
```

将比下列 SQL 更有效率:

```
SELECT * FROM emp E,location L , category C
WHERE E.empID BETWEEN 1000 AND 2000 AND E.catID =
C.catID AND E.locn = L.locn
```

3.4.4 对影响连接的 init.ora 参数进行优化

对这一部分不实际进行介绍,有兴趣的读者可以参考文献[5]。

(下转第 98 页)

的方向应该就是与 ΔV_x 同向。最后的合力的方向应该是人工势场的合力与修正力的合力方向。最后可以得到合力的定义：

$$F_{合力'} = F_{合力} + F_{mod} \tag{9}$$

当机器人需要向左做出规避动作时，需要对机器人施加一个向 x 轴负方向的修正力，以使得机器人摆脱障碍物。可以看到，在传统的人工势场中，障碍物的排斥力和目标物的吸引力的合力的方向仍然使得机器人做出向右规避的动作。虽然由于障碍物在 x 轴正方向上的速度优势较大，使得机器人在向右规避的过程中不可能从右侧绕过障碍物。这个时候，持续向右的规避动作显然是徒劳的，违背了路径最短的原则。此时，就需要机器人聪明地“分析”一下当前的形势，做出向左规避的动作。

这里仅仅需要机器人做出这个向左规避的动作，而不需要考虑当前障碍物机器人具体的位置，即不考虑当前的障碍物 i 对机器人的排斥力的大小。这时考虑的修正力的大小就是目标物的引力和除去当前障碍物以外的其他障碍物对该机器人的排斥力之和的合力，这个力的方向指向 x 轴的负方向。公式如下：

$$F_{mod} = -\left(\sum_{j=0,\cdots,n,j\neq 1} F_{rep(i)} + F_{att}\right) n_x \tag{10}$$

$$F_{合力'} = F_{mod} + \sum_{j=0,\cdots,n,j\neq 1} F_{rep(j)} + F_{att} \tag{11}$$

其中 F_{att} 表示目标物对机器人的吸引力；$F_{rep(j)}$ 表示第 j 个障碍物对机器人的排斥力。

综上所述，在每个仿真周期的开始，需要根据障碍物、目标物、机器人的位置以及速度矢量对机器人的下一步的动作做出预判，这样能更好地消除无谓的避障或者说是徒劳的避障。

3 仿真结果

将上述改进的人工势场方法应用于搭建的 Simulator_APF 平台，结果令人满意。如图4～图6所示，在将机器人的自身速度和加速度矢量引入人工势场法后，应用了选择算法的人工势场法能很好地根据当前的机器人、障碍物、目标物的位置对下一步的动作做出预判，分别做出向左规避(图4)、向右规避(图5)以及正常人工势场规避(图6)的动作，能很好地实现动态环境下的障碍物的规避。

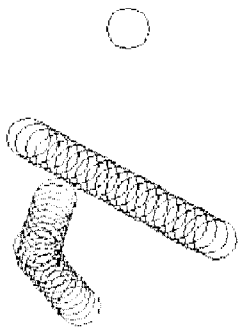

图4 机器人做出向左的规避动作

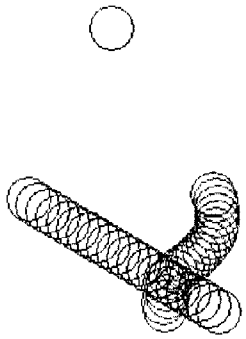

图5 机器人做出向右的规避动作

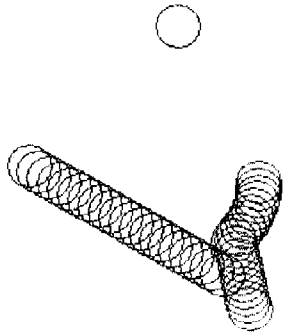

图6 正常人工势场的情况

参考文献：

[1] 张 祺，杨宜民．基于改进人工势场法的足球机器人避碰控制[J]．机器人，2002，24(1)：12－15．

[2] Vadakkepat P, Tan K C, Wang Ming－Liang. Evolutionary Artificial Potential Fields and Their Application in Real Time Robot Path Planning[Z]. Singapore: National University of Singapore, 1998.

[3] Vadakkepat P, Tong Heng, Liu Xin. Application of Evolutionary Artificial Potential Field in Robot Soccer System[Z]. Singapore: National University of Singapore, 2001.

[4] Simmons R, Apfelbaum D, Burgard W, et al. Coordination for Multi－Robot Exploration and Mapping[Z]. [s. l.]: Carnegie Mellon University, 1999.

(上接第95页)

4 结束语

文中对 Oracle9i 中的查询特性进行了分析，给出了在开发过程中如何编写高效查询语句的方案，并着重对影响 SQL 查询性能的表连接进行了分析。开发者在实际应用中，可以以此为参考，避免编写出不良查询语句从而影响系统的性能。当然，Oracle9i 中查询过程是非常复杂的，文中只是对其中比较有影响的关键部分进行了介绍，其它的如远程表的连接、数据仓库中的连接等并没有在文中涉及到。

参考文献：

[1] Whalen E, Schroeter M. Oracle 性能调整与优化[M]. 北京：人民邮电出版社，2002.

[2] Niemiec R J. Oracle 9i 性能调整[M]. 北京：清华大学出版社，2004.

[3] Ault M. Oracle 数据库管理与维护技术手册[M]. 北京：清华大学出版社，2003.

[4] Green C D. Oracle9i Database Performance Tuning Guide and Reference[EB/OL]. http://download－west.oracle.com/docs/cd/B10501－01/server.920/a96533.pdf, 2002.

[5] Lawson C. Oracle 性能优化科学与艺术[M]. 北京：清华大学出版社，2004.