数据库优化设计方法初探
Discussion on Optimization of Database Design

吴 璟莉[1],刘仁辉[2],何冬黎[1]
Wu Jingli[1],Liu Renhui[2],He Dongli[1]

(1. 广西师范大学数学与计算机科学学院,广西桂林 541004;2. 桂林空军学院,广西桂林 541003)
(1. Dept. of Math. & Comp. Sci., Guangxi Normal Univ., Guilin, Guangxi, 541004, China; 2. Guilin Airforce Academy, Guilin, Guangxi, 541003, China)

摘要:从范式优化、索引优化、表的优化及查询优化探讨数据库优化设计的方法,并对它们的综合使用进行分析。数据库优化设计时,在逻辑设计阶段,要根据范式优化及表优化中的要求设计数据库逻辑结构,对两者的利弊进行权衡,选出折衷的方案,既可以避免不一致性和数据异常现象,又不影响整体的性能;在数据库物理设计阶段,要根据索引优化中的要求在有关属性或属性的组合上建立索引,以优化数据库物理结构;在数据库查询阶段,采用在条件子句中使用索引,嵌套查询,迟早执行选择操作,迟早消除不满足条件的记录,分组计数和避免困难的正规表达式6种优化策略,可大大提高查询效率。

关键词:数据库 优化设计 范式 索引 临时表 查询
中图法分类号:TP311 **文献标识码**:A **文章编号**:1002-7378(2005)02-0118-04

Abstract: Four methods of database design optimizing are discussed in standardizing relation pattern, constituting index and temporary table, optimizing query semanteme. At the stage of logical design, logical structure has to be designed based on the demands of normal form and table optimizings. A moderate scheme is chosen by the comparison of advantages and disadvantages of these two options to avoid, disaccord and data abnormity. At the stage of database physical design, some indices need to be established on the relative attributes according to the demands of index optimization to optimize database physical structure. Six kinds of query optimization are revealed to improve query efficiency.

Key words: database, design optimizing, normal form, index, temporary table, query

随着数据库技术的发展,人们越来越依赖于计算机化的业务数据。但由于传统的基于文件的数据存储和检索方法不灵活且难于管理,因而数据库优化设计成为人们关注的问题。如何有效地组织和处理大型数据库的海量数据,使得人们能方便、准确、快捷地完成对数据的存取操作,成为对数据库的建设及使用的瓶颈。

运行速度及执行效率是衡量数据库系统性能的重要标准,为了保证系统性能和吞吐量,需要对数据库的逻辑设计和物理设计进行优化,并贯穿于数据库设计的始终,这就是数据库优化设计的首要任务。本文从范式优化、索引优化、表的优化及查询优化四个方面探讨数据库优化设计的方法,并对它们的综合使用进行分析。

1 范式优化

数据库逻辑设计的结果不是惟一的,好的关系模式通常需要满足2个条件:正确表达数据语义和避免数据冗余、异常及不一致问题的出现。设计过程中,要从这2个基本条件出发,不要顾此失彼,这样才能在提高数据库性能的同时保证数据库的正确性。

将模式的范式作为评价关系模式优劣的标准具有一定的科学性。通常一个关系模式达到BCNF或3NF时,被认为具有较好的性能。当某关系模式达

收稿日期:2004-12-07
修回日期:2005-01-07
作者简介:吴 璟莉(1978-),女,满族,广西博白人,助教,主要从事数据库原理及应用、遗传算法、电子商务等研究。

到 BCNF 时，可以有效消除数据冗余和异常现象，但有时不一定保持原关系模式的函数依赖关系，破坏了数据语义。因此在设计时应统筹兼顾，先尽可能设计成 BCNF 模式集，若此时达不到保持函数依赖的特点，则降低范式要求，改成 3NF 模式集，以保证模式集正确地表达数据语义[1]。

2 索引优化

索引是数据库中重要的数据结构，它的根本目的是为了提高查询效率，改善系统性能。但对索引的存储和维护操作的同时会给系统的空间和时间上带来一些负面影响，牺牲一定的系统性能。因此设计时应尽量选择有用的索引，在提高查询速度和节省存储空间之间寻求最佳的平衡点：

(1)在大型关系数据库中，若处理的关系表较小，则无需建立索引。因为数据量较小时，直接扫描便可很快遍历整个表，建立索引反而会加重系统负担。

(2)如果数据的更新比较频繁，维护索引所付出的代价则很大，若这种代价超出了查询上所获得的好处，此时索引不可取。

(3)选择合适的列建立索引对平衡索引的正负面影响有重要作用。通常在有如下特性的数据列建立索引[2]：①定义有主键和外键的数据列；②需要在指定范围中快速或频繁查询的列；③需要按排序顺序快速或频繁检索的列；④在集合过程中需要快速或频繁组合到一起的列。

如有下列情况时则不考虑创建索引[2]：①在查询中几乎不涉及的列；②不同值少的列，比如在学生表的“性别”列上只有“男”与“女”2 个不同值，就无必要建立索引；③由文本、图像等数据类型定义的列。

(4)索引分为聚集索引和非聚集索引，建立索引时，应考虑对两者的选择：①经常作为查询结果排序条件的字段应建为聚集索引。例如查询结果要对年龄排序，可在年龄字段上建立聚集索引；②当以某字段为查询条件，需要回传局部范围的大量数据时，应在此字段上建立聚集索引，而当查询所获得的数据量较少时，有必要在此字段上建立非聚集索引。例如回传某个时间段之间的数据，可考虑在日期字段上建聚集索引；③应在内容重复性较大的字段上建立聚集索引，而当某字段的数据惟一性较高时，有必要建立非聚集索引。例如学生的成绩重复性较大，可考虑在成绩字段上建立聚集索引对索引列和索引类型的选择没有绝对的界限，在实际应用中，要综合各要素点具体分析，以达到系统的性能综合最优。

3 查询优化

在数据库的所有操作中，查询操作占有很大的比例，查询速度的快慢直接影响到应用系统的生命力，因此设计较为优化的查询语句对提高数据库的整体性能有着重要的作用。实践证明，在许多情况下均可找到语义等价的查询，而选择不同的查询语句对查询效率的影响大不相同，尤其是在一些大规模的关系数据库应用系统中更加明显。本文以应用实例为基础，结合数据库理论，探讨查询优化技术在现实系统中的运用。

设在某学生管理系统中有如下 3 个关系：

(1)学生登记表 S(学号 SNO，姓名 SNAME，年龄 AGE，性别 SEX，院系 SDEPT)；

(2)学习登记表 SC(学号 SNO，课程号 CNO，成绩 GRADE)；

(3)课程登记表 C(课程号 CNO，课程名 CNAME，开课院系 CDEPT，教师 TNAME)。

3.1 在条件子句中使用索引[3]

正确使用索引可以大大提高查询效率，在条件子句中应尽量考虑以下有用索引的使用。

(1) 使用单列索引。

例如，在学生登记表中，如果创建学号为单列索引，那么下列查询语句的 WHERE 子句中应使用学号这个索引，使之成为有用索引。如果使用了其他字段，创建学号这个索引则为无用索引：

```
SELECT  SNO,SNAME,SEX
FROM  S
WHERE  SNO='S1'
```

(2)使用复合索引时，必须保证在条件子句中首先使用复合索引的第一列。

例如，在学习登记表中，如果创建学号和课程号为复合索引，下列查询中复合索引的使用是有用的，因为 SNO 是复合索引的第一列字段：

```
SELECT  SNO,GRADE
FROM  SC
WHERE  SNO='S3'  AND  CNO='C1'
```

但是，下列复合索引的使用是没用的，系统仍然采用顺序扫描方式：

```
SELECT  SNO, GRADE
FROM  SC
WHERE  CNO='C1'  AND  SNO='S3'
```

3.2 采用嵌套查询

嵌套查询(子查询)是指在WHERE或HAVING条件子句中又包含了另一个SELECT查询语句的查询。执行时先执行最内层的子查询,再由内到外逐层进行,通过子查询的层层选择、投影,中间结果元组及属性的数目都得到有效的缩减,使条件判断时的扫描工作仅限于较小范围内,提高检索效率。嵌套查询比联接查询效率要高。

例如,查询选修课程名为"数据库原理"的学生学号和姓名。

```
查询1:SELECT SNO,SNAME FROM C,SC,S
      WHERE CNAME='数据库原理'AND C.
      CNO=SC.CNO
      AND SC.SNO=S.SNO
查询2:SELECT SNO,SNAME FROM S
      WHERE SNO IN
      (SELECT SNO FROM SC WHERE CNO IN
      (SELECT CNO FROM C WHERE CNAME=
      '数据库原理'))
```

由于查询1采用联接查询,只有学号和姓名为输出属性,但几乎所有属性都要参加选择、联接运算,多余的属性占用了较大的内存空间;另外3个表的笛卡尔积将产生大量的中间结果,使选择操作时需处理大量数目的元组。

查询2采用子查询,投影和选择运算同时进行,对C表进行扫描选择后,仅剩下少量满足条件的元组,接着投影操作将其它属性统统剔除,只保留"课程号"一个属性,这样产生一个元组数目和属性数目都较小的中间结果,大大减少了其外层对SC表查询时的比较次数;同理,对SC表的扫描结果又减少了其外层对S表查询时的比较次数。从而提高了检索效率。

这里需要指出的是,使用嵌套查询时,要避免采用相关子查询。即子查询中查询条件依赖于外层查询中的某个值,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,比联接查询的效率更低。

3.3 尽早执行选择操作

尽可能早地执行选择操作,以减少运算量。例如,检索选修课程名为"高数"的学生的学号和姓名。

```
查询1:SELECT S.SNO,S.SNAME
      FROM S,C,SC
      WHERE C.CNO=SC.CNO AND SC.
      SNO=S.SNO AND CNAME='高数'
查询2:SELECT S.SNO,S.SNAME
      FROM S,C,SC
      WHERE CNAME='高数' AND C.CNO
      =SC.CNO AND SC.SNO=S.SNO
```

执行这两条查询语句时,在对表S、C和SC笛卡尔积的中间结果筛选时,查询2执行的比较次数较查询1要少,因为其首先利用是否选修高数这一条件排除了大量的元组,只有满足此条件的元组才需要进行课程号和学号的相等判断。

3.4 尽早消掉不满足条件的记录

having子句中的限制条件应尽量多地放入WHERE子句中,以尽早把不满足条件的记录消掉。

例如,分别统计选修C1和C2课程的学生人数,当选修超过50人时,显示学生的学号。

```
SELECT SNO FROM SC
GROUP BY CNO
HAVING (CNO='C1' OR CNO='C2') AND
COUNT(*)>50
```

应改为:

```
SELECT SNO FROM SC
WHERE CNO='C1' OR CNO='C2'
GROUP BY CNO
HAVING COUNT(*)>50
```

3.5 分组计数

例如,找出选修全部课程的学生姓名。

```
查询1:SELECT SNAME FROM S
      WHERE NOT EXISTS
        (SELECT * FROM C WHERE NOT
        EXESTS
      (SELECT * FROM SC WHERE S.SNO=
      SC.SNO AND C.CNO=SC.CNO))
查询2:SELECT SNAME FROM S WHERE SNO
      IN
          (SELECT DISTINCT SNO FROM SC
          GROUP BY SNO
          HAVING COUNT(CNO)=(SELECT
          COUNT(CNO) FROM C))
```

由于全部课程构成1个集合,如果成绩登记表按学号分组,只要比较每个学生选修的课程集合是否与全部课程构成的集合相等,即可判断某学生是否选修了全部课程。可以证明,对于有引用关系的集合而言,集合中元素数目相等则集合相同,因此查询条件中若某学生选的课程数目等于课程表的记录总数时,表示该同学选修了全部课程。查询2较查询1

效率较高。

3.6 避免困难的正规表达式

正规表达式即为使用 LIKE 等关键字支持通配符匹配，但这种匹配特别耗费时间。例如，从学生登记表中选出以‘2000’为前缀的学号的学生。

查询 1：SELECT * FROM S WHERE SNO LIKE '2000 ___'

查询 2：SELECT * FROM S WHERE (SNO > '2000000')AND (SNO < '2001000')

对于这两种查询，既使在学号字段上建立索引，查询 1 仍是采用顺序扫描的方式，查询 2 则是利用学号索引进行定位，显然大大提高了速度。

4 表的优化

数据库逻辑设计时，依据范化理论把数据划分成多个相关的表。但随着规范化程度的增加，查询时要求联接的表的数目和复杂性也随之增加，系统复杂的联接运算将影响整体性能。另外对于某些查询要求，使用单个查询语句获得查询结果是较困难的工作，特别是对于大型数据库系统，不容易检查查询结果，要确定查询的正确与否非常困难。针对这些情况，通过引入临时表来简化查询。

例如，查询某个学生某门课程的成绩。这个查询的使用频率较高，则对其建立一个临时表 TEMPSC（学号（SNO）、姓名（SNAME）、课程名（CNAME）、教师（TNAME）、成绩（GRADE）），并将对表 S、C 及 SC 的查询结果写入表 TEMPSC 中。后续查询时直接对表 TEMPSC 操作即可，简化了查询工作。

又如查询具有最多女生的系的名称。使用单条查询语句获得查询结果较为困难，则可建立临时表 TEMPW（院系（SDEPT）、人数（NUMBER）），先将各院系女生人数的统计结果写入此表，再在表 TEMPW 中查出人数最多的院系名称。通过分解操作过程，使解决办法得以简化。

使用临时表时要注意对它的更新操作，以保持与原始表之间数据的一致性。使用完毕后，应对其删除，释放其所占用的空间。

5 结束语

数据库的优化设计工作对提高系统执行效率起着重要的作用，本文从关系模式的规范化，索引和临时表的建立以及查询优化这几个重要技术的几个方面对数据库优化方案进行探讨。实际运用时，首先在逻辑设计阶段根据范式优化及表优化中的要求设计数据库逻辑结构，对两者的利弊进行权衡，选出折衷的方案，既避免不一致性和数据异常现象，又不影响整体的性能；接着在数据库物理设计阶段根据索引优化中的要求在有关属性或属性的组合上建立索引，以优化数据库物理结构；最后，在数据库查询阶段，采用本文给出，6 种优化策略，可大大提高查询效率。总之，数据库设计时，要根据具体情况将上述几个方面的优化策略有机地结合起来，尽可能使系统效率达到最优。

参考文献：

[1] 丁宝康，董健全编著. 数据库实用教程[M]. 北京：清华大学出版社，2001.

[2] 黄维通编著. SQL Server 2000 简明教程[M]. 北京：清华大学出版社，2002.

[3] 陈和平，李 军. 用电管理信息系统的数据库设计与优化[J]. 计算机工程，2000，26(9)：154.

（责任编辑：黎贞崇）

稻瘟病菌的基因组序列测定

现在，世界上最主要的作物之一——水稻的最具破坏性的病原体的基因组序列已经测定完成了。稻瘟病菌是第一种被测定的真菌性植物病原体基因组，由于水稻的基因组已经被测序，所以它为研究寄主与病原体之间的关系提供了一个独特的机会。本研究早期的发现包括一个家族的 G-蛋白，与参与破坏寄主防卫系统的受体耦合在一起；一个对这种病原体有特效的杀菌剂的候选目标。该基因组在过去曾被其他基因元素入侵，很可能有助于在遇到新引进的有抗病能力的水稻品种时迅速发生演化。

据《科学时报》