MySQL 中 InnoDB 引擎索引树的搜索策略

Search Strategy of Index Tree of InnoDB Engine in MySQL

罗 凡[1] 彭秀增[1] 申 春[1] 李肃义[2]

(吉林大学计算机科学与技术学院 长春 130025)[1] (吉林大学仪器科学与电气工程学院 长春 130020)[2]

Abstract Based on the introduction of index structure of InnoDB engine in MySQL, the hash search strategy and the binary search strategy adopted in InnoDB Engine are analyzed deeply and respectively, then the advantages and the disadvantages of index system in InnoDB Engine are summarized.

Keywords Clustered index, Non-clustered index, Node pointer, Hash search, Dinary search

1 引言

MySQL 是 GNU 软件(即开源自由软件)中非常优秀的数据库系统,它完全符合 SQL-92 入门级和 ODBC3.0(Open DataBase Connectivity)规范,在符合 POSIX(Portable Operating System Interface)规范的操作系统上实现了非常高效的关系型数据库管理系统。在 MySQL 4.0 以后,嵌入了芬兰 InnoBase Oy 公司开发的数据存储引擎 InnoDB,该引擎支持事务执行功能。MySQL[1]公司 2004 年 1 月发表了 MySQL 5.0,其内嵌的数据引擎 InnoDB4.01 已经具备了一个大型数据库所需的所有基本性能。本文对 InnoDB 引擎的索引系统进行了研究,以便了解通用型数据库的具体实现技术,为开发具有自主知识产权的国产数据库提供必要的技术支持。

2 InnoDB 索引系统概述

InnoDB 有两种重要的索引类型:聚簇索引[2]和非聚簇索引。

每个 InnoDB 表都有一个被称为聚簇索引的特殊索引来保存记录行信息。如果一个表定义了一个主键(PRIMARY KEY),那么主键的索引就是聚簇索引;如果一个表没有定义主键,那么 MySQL 就选出这个表中第一个具有唯一键值的非空字段做为主键,InnoDB 就用这个键的索引作为聚簇索引;如果表中没有这样的键,InnoDB 将在内部产生一个聚簇索引,它是由 InnoDB 分配给它们的行 ID(row id)按顺序排序的记录行组成。这个行 ID 是一个单调递增并将被插入到新行的 6 字节字段,因而由行 ID 排序的记录顺序也就是插入时的物理顺序。

通过聚簇索引访问一个记录行非常快,这是因为记录行数据与查找到它的索引在同一个页面上。在 InnoDB 中,对于非聚簇索引(也称为辅助索引)记录,它包含相应的聚簇索引主键值。由于 InnoDB 的非聚簇索引结构并不存储实际记录,而是指向相应的聚簇索引,因此 InnoDB 就使用主键值在聚簇索引中查找这条记录。需要注意的是,如果主键太长,那么辅助索引将会占用更多的存储空间。

无论是聚簇索引,还是非聚簇索引,在内存中都对应一棵索引树 B-tree[3]。

2.1 B-tree 的结构

B-tree 的叶结点页(层 0)存放索引记录,在层 n(n>0)的页中,存放指向 n-1 层的结点指针。对于每个页,只有一个对应的结点指针。因此,InnoDB 中采用的是普通的 B-tree(父子仅有单向指针),而不是 B-link tree(父子有双向指针)。

结点指针包含一条索引记录的前缀 P[4],前缀具有足够的长度,通过它可以唯一确定一条索引记录,最后一个字段的内容就是子结点页的页号。子结点页可以存储结点字母顺序上大于等于 P 小于 P1(如果在父结点页中还存在下一个结点指针,P1 是其前缀)的结点指针或者索引记录。

如果前缀为 P 的结点指针指向一个非叶结点,那么这个非叶结点页中的最左侧的记录必须具有相同的前缀 P;如果这个结点指针指向叶结点,那么叶结点页中不需要一定包含前缀等于 P 的记录。叶结点的这种设计,可以允许任意删除其中的记录,而不需要修正上一层的结点指针。

树的每一层(0 层除外)都定义一个特殊的记录——minimum record,以任何字母顺序排序,都将它作为最小记录。最小记录通过设置用户记录头中的一个标志位来定义,它的作用是作为一个结点指针的前缀来指向下一层的最左子结点。

综上,索引树的叶结点上存放的是若干个索引记录,非叶结点上存放的是键值和子结点的页号构成的若干个结点指针。其中,键值由索引记录的索引字段组成。一个磁盘页上的所有记录在 B-tree 上的级别相同,即 B-tree 上的一个结点对应一个磁盘页。B-tree 的每层的结点都用双向链表链接,故 B-tree 为 B+-tree(在每页的页头存放前一页的页号和下一页的页号)。

3 InnoDB 中索引树的搜索策略

InnoDB 采用了两种搜索策略:哈西搜索和二进制搜索。

在进行具体搜索操作的过程中,首先尝试进行哈西搜索,如果哈西搜索失败,再尝试进行二进制搜索。

3.1 适应性哈西搜索

如果一个数据库几乎占满了所有主内存,那么在其上进行查询的一个快捷方法就是使用哈西索引。整个 InnoDB 只有一个哈西索引表,不同表的索引通过对应的 B-tree 的 ID 来区分。InnoDB 根据需要,为经常访问的 B-tree 的叶结点页构建哈西索引,哈西表的键值是根据 B-tree 的 ID 以及构成索引的那些字段来构造的。

随着操作的不断尝试,需要实时的调整哈西索引。当数据库表发生变化后,更新哈西索引使用的搜索信息。具体过程如下。

表 1 InnoDB 的搜索模式

搜索模式	功能
PAGE_CUR_G	查找第一个大于待查记录的记录
PAGE_CUR_GE	查找第一个大于等于待查记录的记录
PAGE_CUR_L	查找第一个小于待查记录的记录
PAGE_CUR_LE	查找第一个小于等于待查记录的记录
PAGE_CUR_LE_OR_EXTENDS	SQL 语句含有"column LIKE 'abc%' ORDER BY column DESC"这样的语句

表 2 搜索模式的改变

当前搜索模式	修改后的搜索模式
PAGE_CUR_GE	PAGE_CUR_L
PAGE_CUR_G	PAGE_CUR_LE
PAGE_CUR_LE	PAGE_CUR_LE
PAGE_CUR_L	PAGE_CUR_L
PAGE_CUR_LE_OR_EXTENDS	PAGE_CUR_LE_OR_EXTENDS

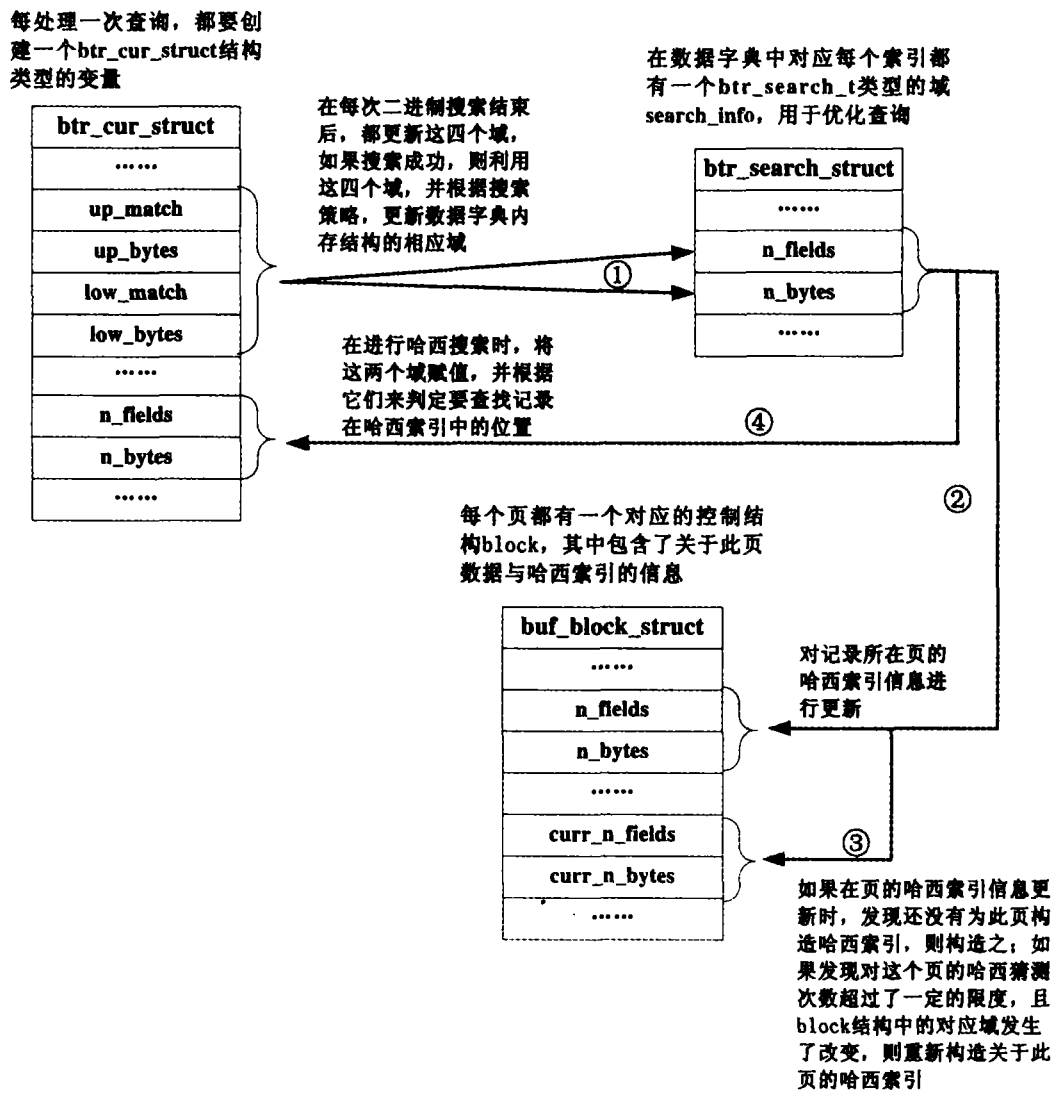

图 1 哈西索引的更新

首先判断当前的索引类型,如果当前的索引类型是非聚簇索引,由于其记录被删除的次数太多,哈西索引不起作用,因此不对哈西索引做任何修改;如果当前的索引类型是聚簇索引,则分以下两种情况

分别进行处理：

第一种情况：

如果采用推荐的哈西信息，实现了成功搜索，则不需要调整哈西索引。

第二种情况：

满足下列条件之一，需要更新哈西信息：

1. 如果到目前为止，采用哈西搜索还没有一次成功，则需要更新哈西信息。

2. 搜索信息中包含的信息与游标中的最新搜索信息不匹配，则需要更新哈西信息。

更新哈西信息如图 1 所示。

3.2 二进制搜索

InnoDB 二进制搜索的总体原则：从树根开始，遍历非叶结点页上符合条件的结点指针，并根据这个指针定位下一层的页号，直到找到叶结点页内的相应记录。

页内记录的定位方法是：先定位槽，再定位槽内的记录；定位槽时，采用二分法；槽内的记录则采用顺序比较方法。

通常，InnoDB 的搜索模式为表 1，如果当前搜索的是非叶结点，则需要改变搜索模式为表 2，到达叶结点页之后，恢复原来的搜索模式。

结论 综上，InnoDB 引擎索引系统的优缺点如下：

其优点是在 InnoDB 中通过聚簇索引访问一个记录行非常快，因为记录行数据与查找到它的索引在同一个页面上，而在大多数的数据库系统中，记录行数据与索引记录通常并不是在同一个页面上。在一个表非常大的情况下，这种聚簇索引体系通常比传统的方式更能减少磁盘 I/O。

此外，InnoDB 索引处理系统采用的哈西搜索策略是一种动态地自适应的搜索技术，由于它对于调进主存的页都自动建立了相应的哈西搜索结构，这样就提高了查找速度。通过这个适应性的哈西搜索策略，InnoDB 使它自己更适合于大的主存，更接近于主存数据库系统。

其缺点是在 InnoDB 索引处理系统中仅仅提供了 B 树一种索引结构，哈西在这里只是一种辅助的索引结构。但在许多其它数据库中不只提供一种索引结构，例如：开源数据库 PostgreSQL 就提供了 B 树、R 树和哈西三种索引结构。

参 考 文 献

1 Paul DuBois 著. 杨涛，杨晓云，王群译. MySQL 权威指南[M]. 机械工业出版社，2004，1

2 SilberSchatz A，Korth H F，Sudarshan S 著. 杨冬青，唐世谓译. 数据库系统概念[M]. 机械工业出版社，2003，3

3 Grarcia H，Melia J D，Widom U J 著. 杨冬青，唐世谓，徐其钧译. 数据库系统实现[M]. 机械工业出版社，2001，3

4 Gray J，Reuter A 著. 孟小峰，于戈等译. 事务处理概念与技术[M]. 机械工业出版社，2004，1

(上接第 389 页)

到 $la1$ 的父标签，然后继续将父标签与 l 比较；若不相同则将栈顶元素弹出，转 step4

若栈顶元素为 l^+：比较 $la1$ 与 l。若相同则处理方法同 l^* 时一致；若不同则转 step6

若栈顶元素为 $l?$：比较 $la1$ 与 l。若相同则弹出栈顶元素，判断 $la1$ 的父标签偏移量是否为 0，为 0 转 step5，不为 0 则依据父标签偏移量找到 $la1$ 的父标签，然后转 step4；若与栈顶元素不同，则弹出栈顶元素

};

step5：

判断栈 W 是否为空，若不为空则转 step6，为空则表示匹配了查询路径，将该匹配词 w 的出现频率 tf 的值累加入动态查询结果存储器的对应项内，即标记为〈文档号，k〉的项；

step6：

此匹配词不符合查询路径表达式，该项推出运算；

$k=k-1$;

};

};

END;

结束语 未来 XML 文档数量的日益增多，使得人们对专业的 XML 搜索引擎的需要越来越迫切。本文给出了一个索引并查询 XML 文档的方法，这是 XML 搜索引擎技术中的重要组成部分。本文提出的查询条件的输入格式，客观上满足了搜索引擎界面层层递进引导用户进行查询搜索的需要；定义的索引格式也充分考虑了磁盘空间和查询时的检索速度等诸方面因素。

参 考 文 献

1 Wilkinson R. Effective Retrieval of Structrured Documents. In: Proc. the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, 1994. 311～317

2 NIST, Guide to Z39. 50/PRISE 1. 0, NIST document, 1995

3 Shin D, Jang H, Jin H. BUS: An effective Indexing and Retrieval cheme in Structured Documents. ACM, 1998