**浅谈高职院校数据库原理中索引的研究与讲解**

谢晓伟

【摘要】数据库原理是计算机院系的一本基础专业课，作为高职院校则更加注重于数据库技术的应用，本文作者长期从事高职院校数据库原理的教学工作，本文重点讲解了数据库原理中“索引”的概念及用法，有助于数据库管理及程序开发的性能调优。

【关键词】数据库 索引 B-Tree

**Abstract：**The principle of database is a basic professional course in computer department. As a higher vocational college, it pays more attention to the application of database technology. The author has been engaged in the teaching of database development technology in Higher Vocational Colleges for a long time. This paper focuses on the concept and usage of "index" in the principle of database, which helps to optimize the performance of database management and program development.

**Key Words：**database index B-Tree

**0 引言**

数据库系统的应用给人们的数据处理提供了一种高效、便捷的方式，但对于以亿级数据量计算的大型关系型数据库，如何有效地应用索引就成了提高数据库系统查询时效率的关键。

**1 索引的概念**

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。通俗来讲，数据库系统类似于我们生活中常用的字典，而索引就等同于字典的目录，我们要从一本字典中查找某个汉字，如果没有目录的话，意味着要将字典从头翻到尾逐字去找，这样很浪费时间，而有了目录后，我们可以根据目录中该汉字的页数再到字典中去找到这个汉字，这样会快很多。

**2 索引的优缺点**

索引的优点是可以快速进行数据表的检索，减少I/O次数，提高数据检索效率，根据索引分组和排序，可以加快分组和排序操作。

索引的缺点是，索引会占用存储空间，一般来说，索引表占用的空间的数据表的1.5倍，而构建索引的同时会降低数据表的修改操作（删除，添加，修改）的效率，另外索引表的维护和创建需要时间成本，这个成本随着数据量增大而增大。

**3 索引的实现原理**

不同的数据库系统所采用的索引实现原理不同，常见的实现原理有：哈希索引、全文索引、BTree索引和B+Tree索引等。

**3.1 哈希索引**

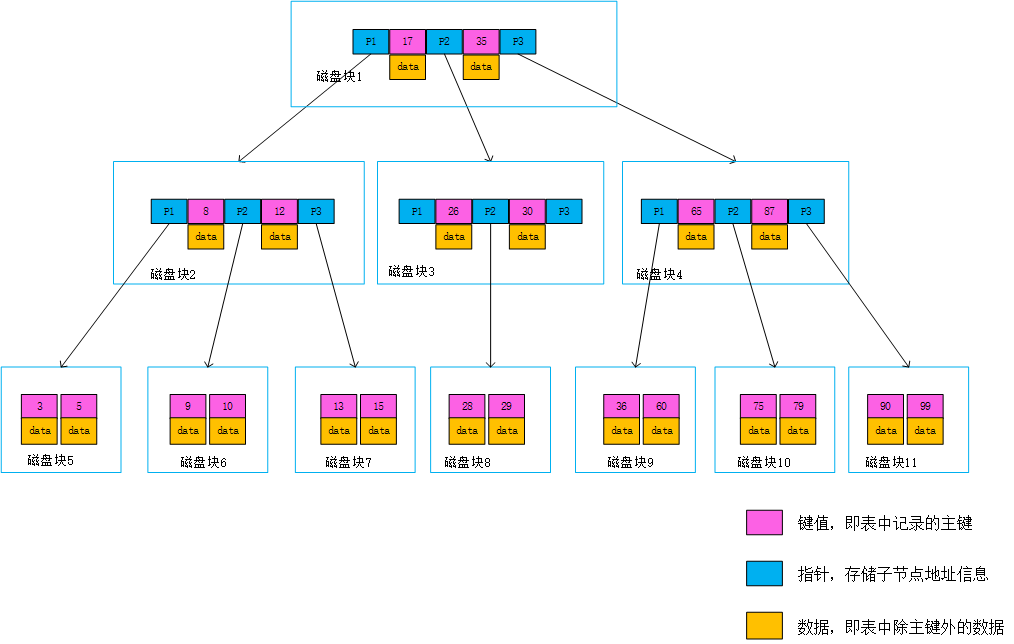
哈希索引用索引列的值计算该值的hashCode，然后在hashCode相应的位置存执该值所在行数据的物理位置，因为使用散列算法，因此访问速度非常快，但是一个值只能对应一个hashCode，而且是散列的分布方式，因此哈希索引不支持范围查找和排序的功能。

**3.2 全文索引**

对于文本的大对象，或者较大的CHAR类型的数据，如果使用普通索引，那么匹配文本前几个字符还是可行的，但是想要匹配文本中间的几个单词，那么就要使用LIKE %word%来匹配，这样需要很长的时间来处理，响应时间会大大增加，这种情况，应该使用全文索引，在生成全文索引时，会为文本生成一份单词的清单，在检索时根据这个单词的清单来检索，从而提高了检索效率。

**3.3 B-Tree索引**

B-Tree是平衡多路查找树，如下图所示：

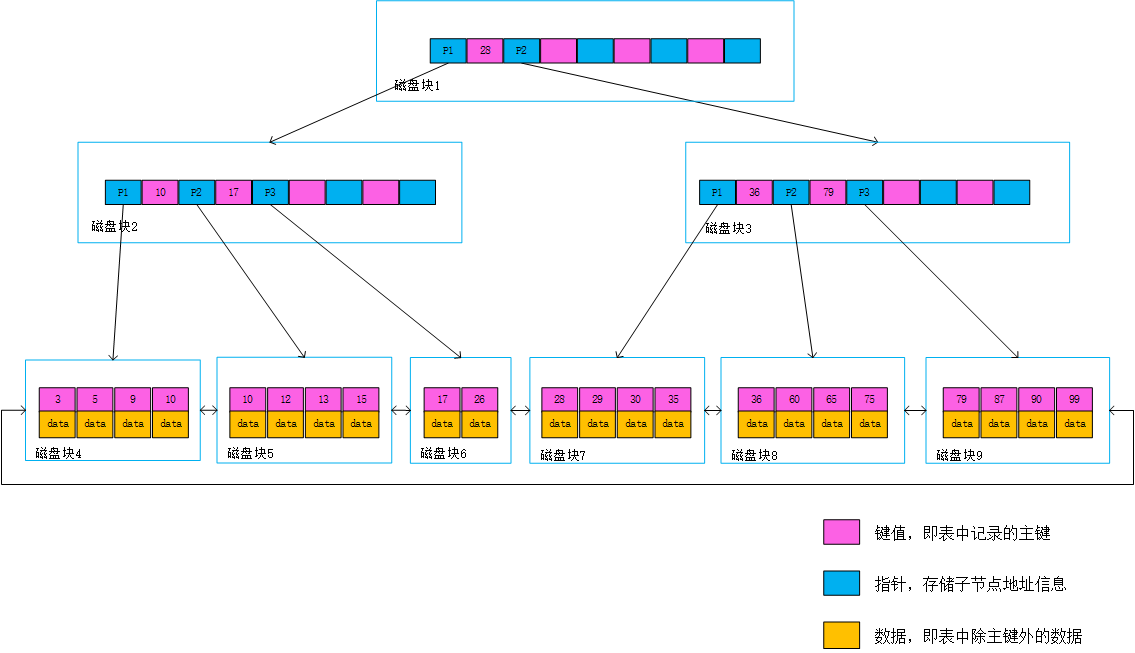


上图中，每个节点占用一个盘块的磁盘空间，一个节点上有两个升序排序的关键字和三个指向子树根节点的指针，指针存储的是子节点所在磁盘块的地址；两个关键词划分成的三个范围域对应三个指针指向的子树的数据的范围域。以根节点为例，关键字为17和35，P1指针指向的子树的数据范围为小于17，P2指针指向的子树的数据范围为17~35，P3指针指向的子树的数据范围为大于35。

针对B-Tree索引，模拟寻找关键字29的过程，首先会把磁盘块1由磁盘加载到内存，此时发生一次I/O开销，在内存中用二分查找确定29在17和35之间，锁定磁盘块1的P2指针，再通过磁盘块1的P2指针所指向的磁盘地址把磁盘块3由磁盘加载到内存，发生第二次I/O开销，29在26和30之间，锁定磁盘块3的P2指针，通过指针所指向的磁盘地址加载磁盘块8到内存，发生第三次I/O开销，同时内存中做二分查找找到29，结束查询，总计三次I/O开销。在实际应用过程中，3层的B-Tree可以表示上百万的数据，如果上百万的数据查找只需要三次I/O开销，性能提高将是巨大的，如果没有索引，每个数据项都要发生一次I/O开销，那么总共需要百万次的开销，显然时间成本非常非常高。

**3.4 B+Tree索引**

B+Tree是在B-Tree基础上的一种优化，所有数据记录节点都是按照键值大小顺序存放在同一层的叶子节点上，而非叶子节点上只存储key值信息，这样可以大大加大每个节点存储的key值数量，降低B+Tree的高度，使其更适合实现外存储索引结构，如下图所示：



通常在B+Tree上有两个头指针，一个指向根节点，另一个指向关键字最小的叶子节点，而且所有叶子节点（即数据节点）之间是一种链式环结构。因此可以对B+Tree进行两种查找运算：一种是对于主键的范围查找和分页查找，另一种是从根节点开始，进行随机查找。

**4 创建索引**

CREATE UNIUQE | BITMAP INDEX <schema>.<index\_name>

ON <schema>.<table\_name>

(<column\_name> | <expression> ASC | DESC,

<column\_name> | <expression> ASC | DESC,...)

TABLESPACE <tablespace\_name>

STORAGE <storage\_settings>

LOGGING | NOLOGGING

COMPUTE STATISTICS

NOCOMPRESS | COMPRESS<nn>

NOSORT | REVERSE

PARTITION | GLOBAL PARTITION<partition\_setting>

UNIQUE | BITMAP：指定UNIQUE为唯一值索引，BITMAP为位图索引，省略为B-Tree索引。

<column\_name> | <expression> ASC | DESC：可以对多列进行联合索引，当为expression时即“基于函数的索引”

TABLESPACE：指定存放索引的表空间(索引和原表不在一个表空间时效率更高)

STORAGE：可进一步设置表空间的存储参数

LOGGING | NOLOGGING：是否对索引产生重做日志(对大表尽量使用NOLOGGING来减少占用空间并提高效率)

COMPUTE STATISTICS：创建新索引时收集统计信息

NOCOMPRESS | COMPRESS<nn>：是否使用“键压缩”(使用键压缩可以删除一个键列中出现的重复值)

NOSORT | REVERSE：NOSORT表示与表中相同的顺序创建索引，REVERSE表示相反顺序存储索引值

PARTITION | NOPARTITION：可以在分区表和未分区表上对创建的索引进行分区

**5 结束语**

索引作为数据库原理的一个重要知识点，对于数据库的管理和程序开发都具有十分重要的意义，掌握了数据库索引实现的基本原理，才能根据不同的情况创建合理的数据库索引，从而更好地实现数据库系统的应用。

**参考文献：**

[1]李素奇.关于SQL索引建立规则与优化的探讨[J].科技展望, 2014,(19):214-215.

[2]赵光亮, 舒小松.Navicatfor MySQL平台中的SQL语言分析与应用[J].无线互联科技, 2017, (19) :254-256.

[3]郑阿奇.MySQL教程[M].北京:清华大学出版社, 2017:90.

[4]明日科技.SQL Server从入门到精通[M].第2版.北京：清华大学出版社，2017:256-281.

[5]周慧，施乐军，崔玉礼.SQL Server2012数据库技术及应用[M].第4版.北京：人民邮电出版社，2017:133-136.

[6]王利.SQL SERVER数据库性能调整与优化[D].电子科技大学, 2007.