从千万级数据查询来聊一聊索引结构和数据库原理！面试必考！

程序员面试 6月10日

以下文章来源于码大叔，作者码大叔

码大叔
资深程序员、架构师社区。实战踩坑，硬核分享，关注微服务、大数据、架构设计、技...

在日常工作中我们不可避免地会遇到慢SQL问题，比如笔者在之前的公司时会定期收到DBA彪哥发来的Oracle AWR报告，并特别提示我某条sql近阶段执行明显很慢，可能要优化一下等。对于这样的问题通常大家的第一反应就是看看sql是不是写的不合理啊诸如：*"避免使用in和not in，否则可能会导致全表扫描"" 避免在where子句中对字段进行函数操作"*等等，还有一种常见的反应就是这个表有没有加索引？绝大部分情况下，加了个索引基本上就搞定了。

既然题目是《从千万级数据查询来聊一聊索引结构和数据库原理》，首先就来构造一个千万级的表直观感受下。我们创建了一张user表，然后插入了1000万条数据，查询一下：

id	username	password	phone	email	birthday
10000000	码大叔9999999	123456	1880000	madash	2002-05-12

```
select * from user where id = 10000000
```

只读 查询时间: 29.315s

用了近30秒的时间，这还是单表查询，关联查询明显会更让人无法忍受。接下来，我们只是对id增加一个索引，再来验证一把：

id	username	password	phone	email	birthday	address
10000000	码大叔9999999	123456	1880000	madash	2002-05-12	南京路1号

```
select * from user where id = 10000000
```

查询时间: 0.022s

从30s到0.02s，提升了足足1500倍。为什么加了索引之后，速度嗖地一下子就上去了呢？我们从**【索引数据结构】**、**【Mysql原理】**两个方面入手。

一、索引数据结构

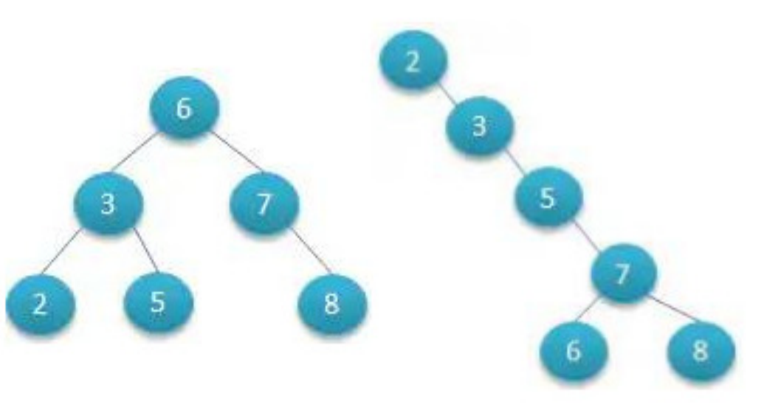
我们先来看下 MySQL官方对索引的定义：

> 索引（Index）是帮助MySQL高效获取数据的数据结构。

这里面有2个关键词：**高效查找、数据结构**。对于数据库来说，查询是我们最主要的使用功能，查询速度肯定是越快越好。最基本的查找是顺序查找，更高效的查找我们很自然会想到二叉树、红黑树、Hash表、BTree等等。

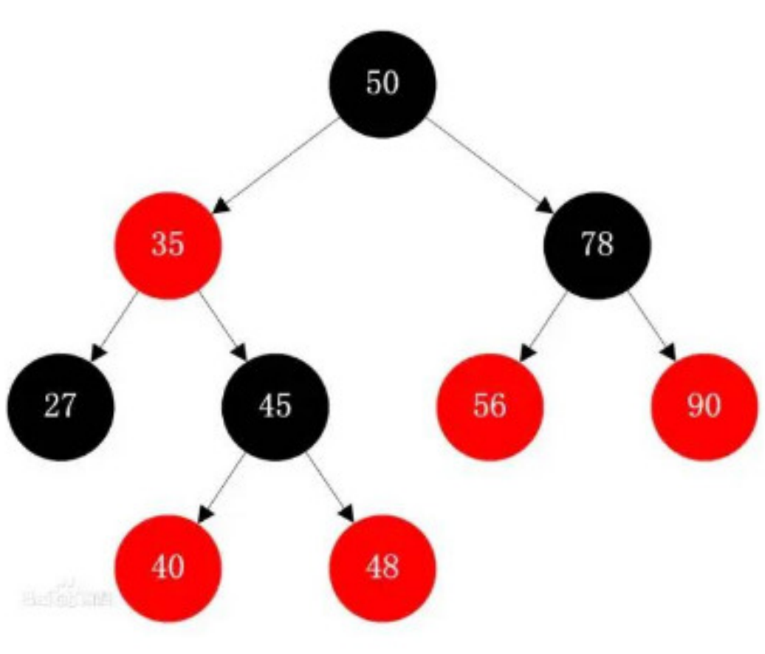
1.1 二叉树

这个大家很熟悉了，他有一个很重要的特点：左边节点的键值小于根的键值，右边节点的键值大于根的键值。比如图1，它确实能明显提高我们的搜索性能。但如果用来作为数据库的索引，明显存在很大的缺陷，但对于图2这种递增的id，存储后索引近似于变成了单边的链表，肯定是不合适的。



1.2 红黑树

也称之为平衡二叉树。在JDK1.8后，HashMap对底层的链表也优化成了红黑树（后续文章我们可以讲讲Hashmap1.8之后的调整）。平衡二叉树的结构使树的结构较好，明显提高查找运算的速度。但是缺陷也同样很明显，插入和删除运算变得复杂化，从而降低了他们的运算速度。对大数据量的支撑很不好，当数据量很大时，树的高度太高，如果查找的数据是叶子节点，依然会超级慢。



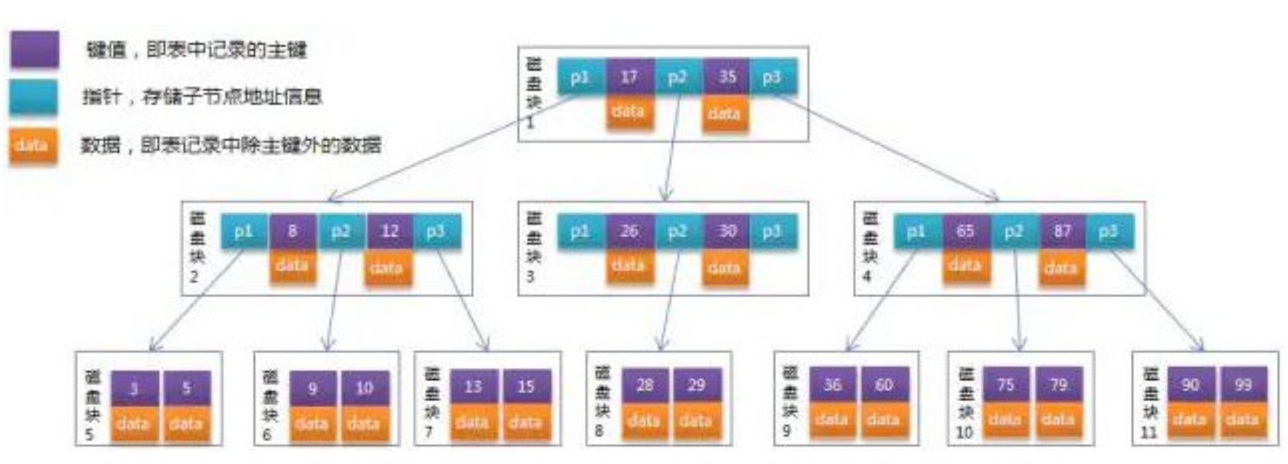
1.3 BTree

B-Tree是为磁盘等外存储设备设计的一种平衡查找树。系统从磁盘读取数据到内存时是以磁盘块（block）为基本单位的，位于同一个磁盘块中的数据会被一次性读取到内存中。在Mysql存储引擎中有页（Page）的概念，页是其磁盘管理的最小单位。Mysql存储引擎中默认每个页的大小为16KB，查看方式：

```
mysql> show variables like 'innodb_page_size';
```

Variable_name	Value
Innodb_page_size	16384

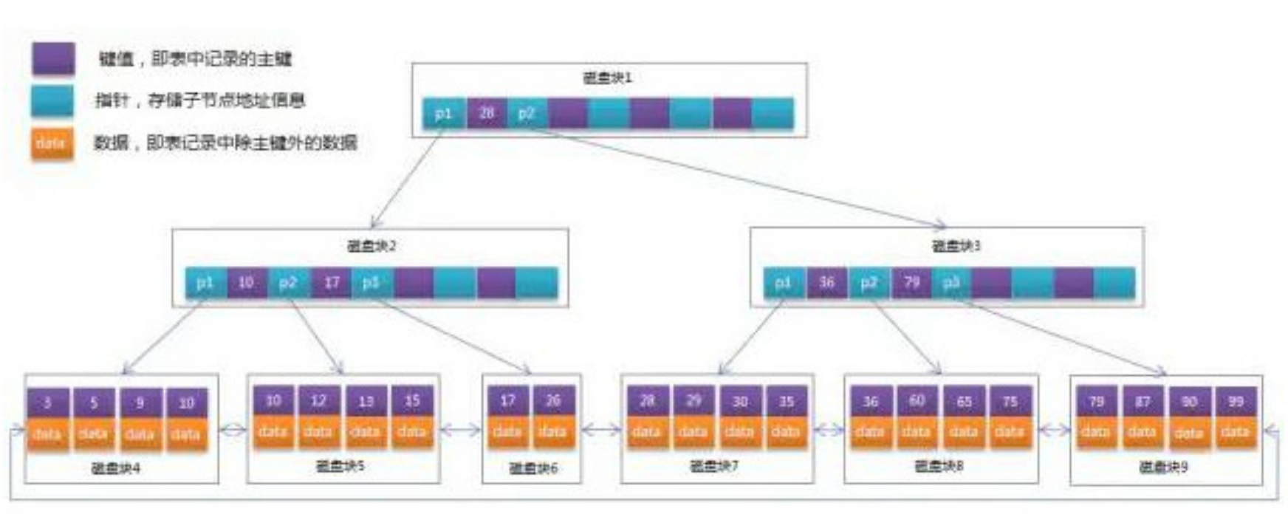
我们也可以将它修改为4K、8K、16K。系统一个磁盘块的存储空间往往没有16K，因此Mysql每次申请磁盘空间时都会将若干地址连续磁盘块来达到页的大小16KB。Mysql在把磁盘数据读入到磁盘时会以页为基本单位，在查询数据时如果一个页中的每条数据都能有助于定位数据记录的位置，这将会减少磁盘I/O次数，提高查询效率。



如上图所示，一棵B树包含有键值、存储子节点的指针信息、及除主键外的数据。相对于普通的树BTree将横向节点的容量变大，从而存储更多的索引。

1.4 B+Tree

在B-Tree的基础上大牛们又研究出了许多变种，其中最常见的是B+Tree，MySQL就普遍使用B+Tree实现其索引结构。



与B-Tree相比，B+Tree做了以下一些改进：
1、非叶子节点，只存储键值信息，这样极大增加了存放索引的数据量。
2、 所有叶子节点之间都有一个链指针。对于区间查询时，不需要再从根节点开始，可直接定位到数据。
3、 数据记录都存放在叶子节点中。根据二叉树的特点，这个是顺序访问指针，提升了区间访问的性能。
通过这样的设计，一张千万级的表最多只需要3次磁盘交互就可以找出数据。

二、Mysql部分原理说明

这一部分我们选举几个日常面试过程中或者使用过程中比较常见的问题通过问答的形式来进行讲解。

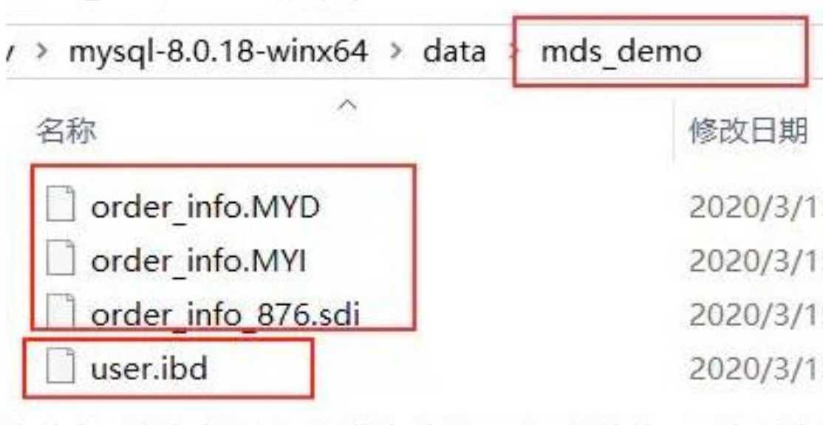
2.1、数据库引擎MyISAM和InnoDB有什么区别

- MyISAM：
  在Mysql8之前，默认引擎是MyISAM，其目标是快速读取。
  特点：
  1、读取非常快，如果频繁插入和更新的话，因为涉及到数据全表锁，效率并不高
  2、保存了数据库行数，执行count时，不需要扫描全表；
  3、不支持数据库事务；
  4、不支持行级锁和外键；
  5、不支持故障恢复。
  6、支持全文检索FullText，压缩索引。
  建议使用场景：
  1、做很多count计算的，（如果count计算后面有where还是会全表扫描）
  2、插入和更新较少，查询比较频繁的
- InnoDB：
  在Mysql8里，默认存储引擎改成了InnoDB。
  特点
  1、支持事务处理、ACID事务特性
  2、实现了SQL标准的四种隔离级别
  3、支持行级锁和外键约束
  4、可以利用事务日志进行数据恢复
  5、不支持FullText类型的索引，没有保存数据库行数，计算count(*)需要全局扫描
  6、支持自动增加列属性auto_increment
  7、最后也是非常重要的一点：InnerDB是为了处理大量数据时的最大性能设计，其CPU效率可能是其他基于磁盘的关系型数据库所不能匹敌的。
  建议使用场景
  1、可靠性高或者必须要求事务处理
  2、表更新和查询相当的频繁，并且表锁定的机会比较大的情况下，指定InnerDB存储引擎。

2.2 表和数据等在Mysql中是如何存储的

我们新建一个数据库mds_demo，里面有两张表：order_info,user

我们找到mysql存放数据的data目录，存在一个mds_demo的文件夹，同时我们也找到了order_info和user的文件。



名称	修改日期
order_info.MYD	2020/3/15
order_info.MYI	2020/3/15
order_info_876.sdi	2020/3/15
user.ibd	2020/3/15

为什么两张表产生了不同的文件呢？原因很简单，因为创建这两张表时使用了不同的引擎

引擎: MyISAM

引擎: InnoDB

- MyISAM引擎在创建表的时候，会创建三个文件
  .MYD文件： 存放表里的数据
  .MYI文件： 存放索引数据
  .sdi文件： Serialized Dictionary Information的缩写。在Mysql5里没有sdi文件，但会有一个FRM文件，用户存放表结构信息。在MySQL8.0中重新设计了数据字典，改为sdi。MyISAM的索引和数据是分开的，并且索引是有压缩的，所以存储文件就会小很多，MyISAM应对错误码导致的数据恢复的速度很快。
- InnerDB引擎在创建表的时候，只有1个文件.ibd，即存放了索引又存放了文件，参见B+Tree。所以它也被称之为聚集索引，即叶子节点包含完整的索引和数据，对应的MyISAM为非聚集索引。
  补充说明一下：存储引擎是针对表的，而不是针对数据库，同一个库的不同的表可以使用不同的引擎。

2.3 为什么InnoDB必须要有主键，并且推荐使用整型的自增主键？

通过上面的讲解这个问题其实已经很清楚了，为了满足MySQL的索引数据结构B+树的特性，必须要有索引作为主键，可以有效提高查询效率。有的童鞋可能会说我创建表的时候可以没有主键啊，这个其实和Oracle的rownum一样，如果不指定主键，InnoDB会从插入的数据中找出不重复的一列作为主键索引，如果没找到不重复的一列，InnoDB会在后台增加一列rowId做为主键索引。所以不如我们自己创建一个主键。

将索引的数据类型是设置为整型，一来占有的磁盘空间或内存空间更少，另一方面整型相对于字符串比较更快速，而字符串需要先转换为ASCII码然后再一个个进行比较的。

参见B+树的图它本质上是多路多叉树，如果主键索引不是自增的，那么后续插入的索引就会引起B+树的其他节点的分裂和重新平衡，影响数据插入的效率，如果是自增主键，只用在尾节点做增加就可以。

最后特别强调一点：不管当前是否有性能要求或者数据量多大，千万不要使用UUID作为索引。

2.4 为什么Mysql存储引擎中默认每个页的大小为16KB?

假设我们一行数据大小为1K，那么一页就能存16条数据，包含指针+数据+索引。假设一行数据大小为1K，那么一页（1个叶子节点）就能存16条数据；对于非叶子节点，假设ID为bigint类型那么长度为8B，指针大小在Innodb源码中为6B，一共就是14B，那么一页里就可以存储16K/14=1170个(主键+指针)，这样一颗高度为3的B+树能存储的数据为：1170*1170*16=2千万级别。所以我们前面1000万的数据只有0.02s。

2.5 HASH算法的使用场景

名	字段	索引类型	索引方法	注释
* id	id			

Hash算法是一种散列算法，就是计算出某个字段的hash，然后存放在对应的地址中，查找数据时只需要1次定位而不像BTree那样从根节点找到叶子节点经过多次IO操作，所以查询效率非常地高。但同样也有很多的弊端，讲一下最重要的两条。
1、很明显hash只支持=、IN等查询，而不支持范围查询
2、 Hash 索引在任何时候都不能避免表扫描。

所以使用时务必注意。

图片：
本文中的部分图片来源于网络，版本归原作者所有。
参考：

https://www.cnblogs.com/vianzhang/p/7922426.html
https://www.cnblogs.com/yangecnu/p/Introduce-B-Tree-and-B-Plus-Tree.html
http://blog.codinglabs.org/articles/theory-of-mysql-index.html
https://tech.meituan.com/2014/06/30/mysql-index.html
https://www.ucloud.cn/yun/110762.html
https://www.cs.usfca.edu/~galles/visualization/BST.html

向图片作者及内容参考的作者表示感谢！