如何理解INNODB索引

innodb任何对于数据的增删查改都是基于索引结构的操作。Innodb的索引采取了B+ 树的数据结构用来组织数据库数据。

B+ TREE

在理解这个问题之前，我们需要先找到数据库存取的主要问题所在。通常计算机的存储系统可以分为主存（即内存）以及辅存（即外存，如硬盘）两种，主存价格贵、读写速度快、容量低，辅存价格便宜、读写速度慢、容量高。通常主存只用于计算机核心的CPU计算，而对于大量数据的存储都在辅存中。

一个磁盘驱动器：

图2

为了在使用辅存价格低廉，容量大的优点同时尽量避免其读写速度慢的问题。Innodb使用B+树来组织数据。

B+树通常拥有很大的分支因子。这可以减小树的高度h，从而有效减少在检索一个记录时对于辅存的读取次数。因为根节点通常长驻内存中，故在B+ TREE中检索一个叶子结点的数据至多只需要访问辅存h次。通常对于一个B+TREE应用，在做大量数据运算时只需将一部分页面的数据保持在主存中以进行快速运算。

B+TREE除了广泛应用于数据库的索引结构，同时大量文件系统例如 [ReiserFS](https://en.wikipedia.org/wiki/ReiserFS)、[NSS](https://en.wikipedia.org/wiki/Novell_Storage_Services)、[XFS](https://en.wikipedia.org/wiki/XFS)、[JFS](https://en.wikipedia.org/wiki/JFS_(file_system))、[ReFS](https://en.wikipedia.org/wiki/ReFS)、[NTFS](https://en.wikipedia.org/wiki/NTFS)等也使用了该数据结构用于部分元数据存储。

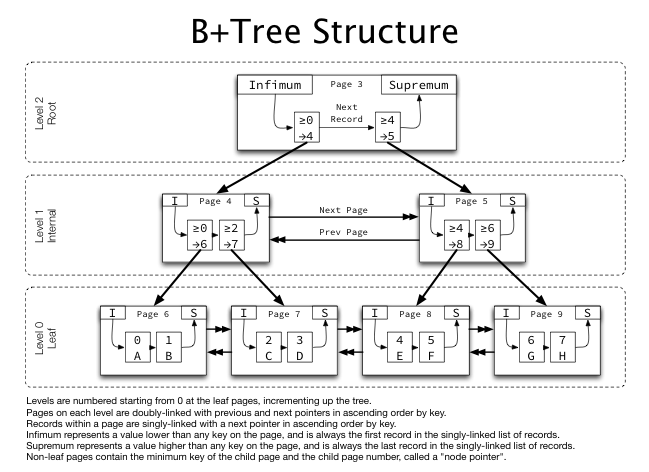
INNODB B+ TREE

B+ TREE的一些重要特性：

1. B+ TREE是分支因子（BRACHING FACTOR：节点的最大子节点个数）很大的平衡树（BALANCE TREE：根节点到任一叶子节点的距离相同）。
2. 若节点可存储的最大关键字个数为N，则该B+ TREE的分支因子b=N+1
3. 除根节点外的节点关键字个数至少为 ⌈b/2 - 1⌉，所有节点最多有b-1个关键字

在innodb中根节点常驻内存中，表查询首先就需要通过datadirectory查询到对应的根节点位置。所有节点都包含了一个逻辑上的最小key值infimum和最大值supremum。在叶子节点上保存了主键key以及行数据。而在非叶子节点中则保存了子节点的最小key值子节点的page编号。所有page的关键字在页内在逻辑上均以单链増序排列（在物理地址上则不是），同高度的page之间则以双链形式连接。

一个简单的B+树。



【图1】

1. 该树的一个节点最多可包含3个key。其分支因子为4。

B+TREE的操作：

1. SERACH：

在这颗B+TREE中查询key为3的数据，则会从根节点开始搜索，沿page3-》4-》7最终在page7中查询到数据值为‘D’。而当我们不使用索引树查询，则直接通过简单的表扫描由page6->7->8->9的顺序遍历最终将在page7中查询到数据值为‘D’。

1. INSERTION
2. DELETION：

注意：由于Innodb的根节点长驻内存并且其page编号也在其创建时便保存在了data directory中，故实际上根节点时无法通过重新移动位置分配空间来完成在容量饱和后的节点分裂动作的。此时将创建一个新的空白page并将原根节点的数据移动至该page并完成分裂。

索引的物理结构

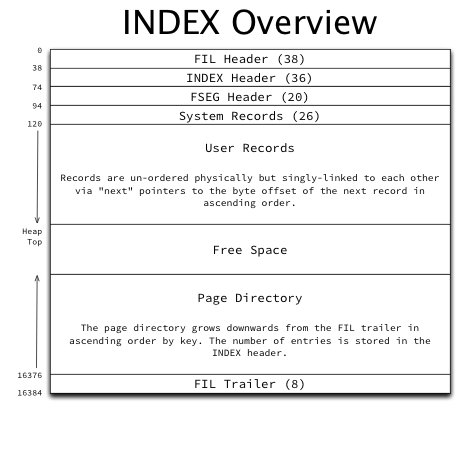
Innodb的数据存储模型

Innodb使用space作为存储模型。一个space 可能包含了一个或多个文件。在MySQL 5.6.6以后，[innodb\_file\_per\_table](https://dev.mysql.com/doc/refman/5.6/en/innodb-parameters.html#sysvar_innodb_file_per_table)默认为开启状态，这意味着创建一张新表则为为该表创建一个.ibd文件，而这个文件实际上是一个完整的space。每一个space都被分配了一个32bit的id作为其引用。而id0总是被分配给innodb的系统空间space0。

Space内部被划分为单个的page。一个page在物理内存中代表着16KB的连续空间（除非在编译时改变了 UNIV\_PAGE\_SIZE 的值或者innodb使用了压缩模式）。每一个page都被分配了一个32bit的页编号pagenumber，也就是space起始位置的偏移量，例如第一页的pagenumber就是0，而第二页偏移量就是 16\*2^10=2^14=16384。（理论上来说innodb的space最大空间为64TiB，这也就是理论上的32bit页编号的最大偏移地址 2^32\*16\*2^10=64\*2^40=64TiB）。每64页（64\*16KB=1MB）为称为一个Extent。

图3

Page可以分为INDEX（索引页,存放节点数据）、SYS（系统页，存放系统相关信息）、INODE（包含许多INODEENTRY节点用于描述FILESGMENT）、TRX\_SYS（事务系统页，包含事务操作相关的信息）等。而对于用户数据的存储和组织主要在INDEX 页中。



FIL Header：所有不同类型的page均具有的头部信息，包含了该page的类型、所属space等信息

INDEX Header：INDEX page的页面信息，包含了页面空间信息、事务信息等

FSEG Header：只有ROOT根节点才使用该Header用于保存所使用的的FSEGMENT指针。 （一个B+TREE结构在INDEX page中由两个FSEGMENT组成，一个用于保存叶子节点，另一个用于保存非叶子节点）

System Records:每个INDEXpage包含两条系统记录，一条为infium，该记录代表一个小于page内的任何key值的key，另一条为supremum，该记录代表一个大于page内的任何key值的key

User Records：实际保存的行数据

Page Directory：对于该page所有key的组织，用于对查询进行二分查找优化

B+TREE

图4