MySQL索引有哪些分类，你真的清楚吗？

〇、MySQL索引都有哪些分类？

按数据结构分类可分为：**B+tree索引、Hash索引、Full-text索引。**
按物理存储分类可分为：**聚簇索引、二级索引（辅助索引）。**
按字段特性分类可分为：**主键索引、普通索引、前缀索引。**
按字段个数分类可分为：**单列索引、联合索引（复合索引、组合索引）。**

一、按数据结构分类

MySQL索引按数据结构分类可分为：**B+tree索引、Hash索引、Full-text索引。**

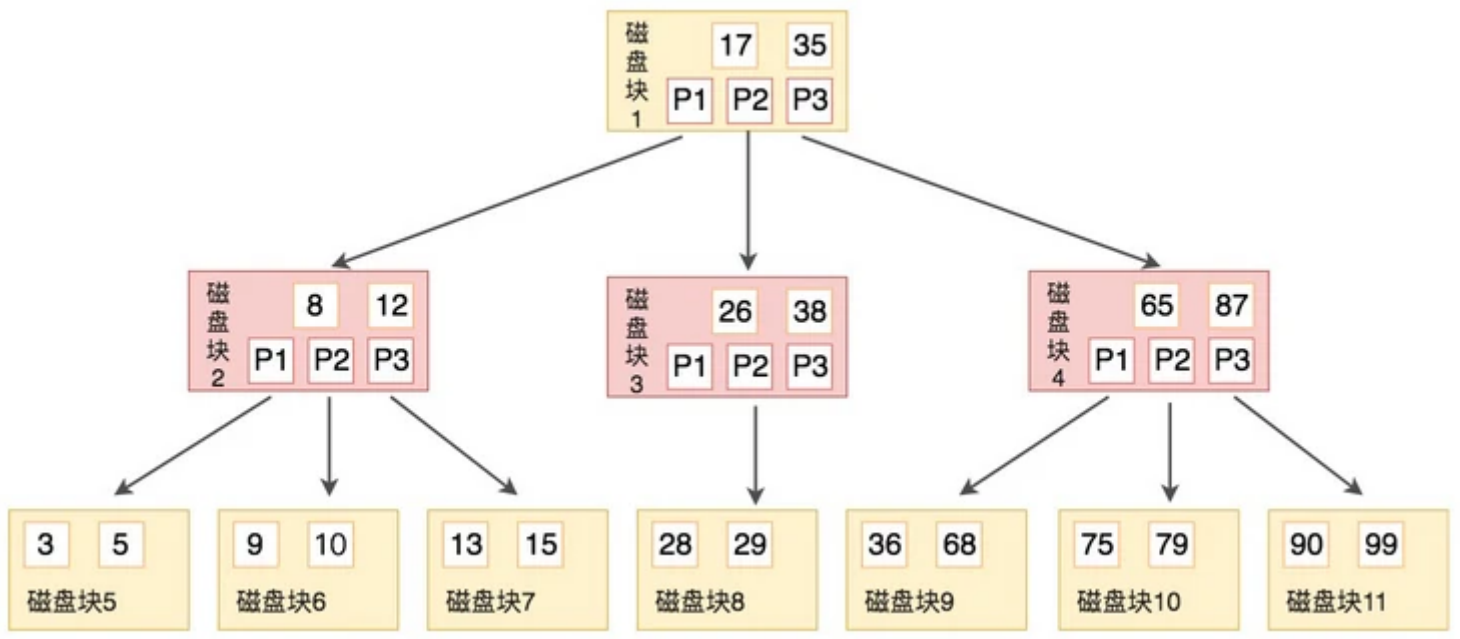
-	InnoDB	MyISAM	Memory
B+tree索引	√	√	√
Hash索引	×	×	√
Full-text索引	√（MySQL5.6+）	√	×

> 注：InnoDB实际上也支持Hash索引，但是InnoDB中Hash索引的创建由存储引擎引擎自动优化创建，不能人为干预是否为表创建Hash索引

B+tree 是MySQL中被存储引擎采用最多的索引类型。**B+tree** 中的 B 代表平衡（balance），而不是二叉（binary），因为 **B+tree** 是从最早的平衡二叉树演化而来的。下面展示B+tree数据结构与其他数据结构的对比。

1. B+tree与B-tree的对比

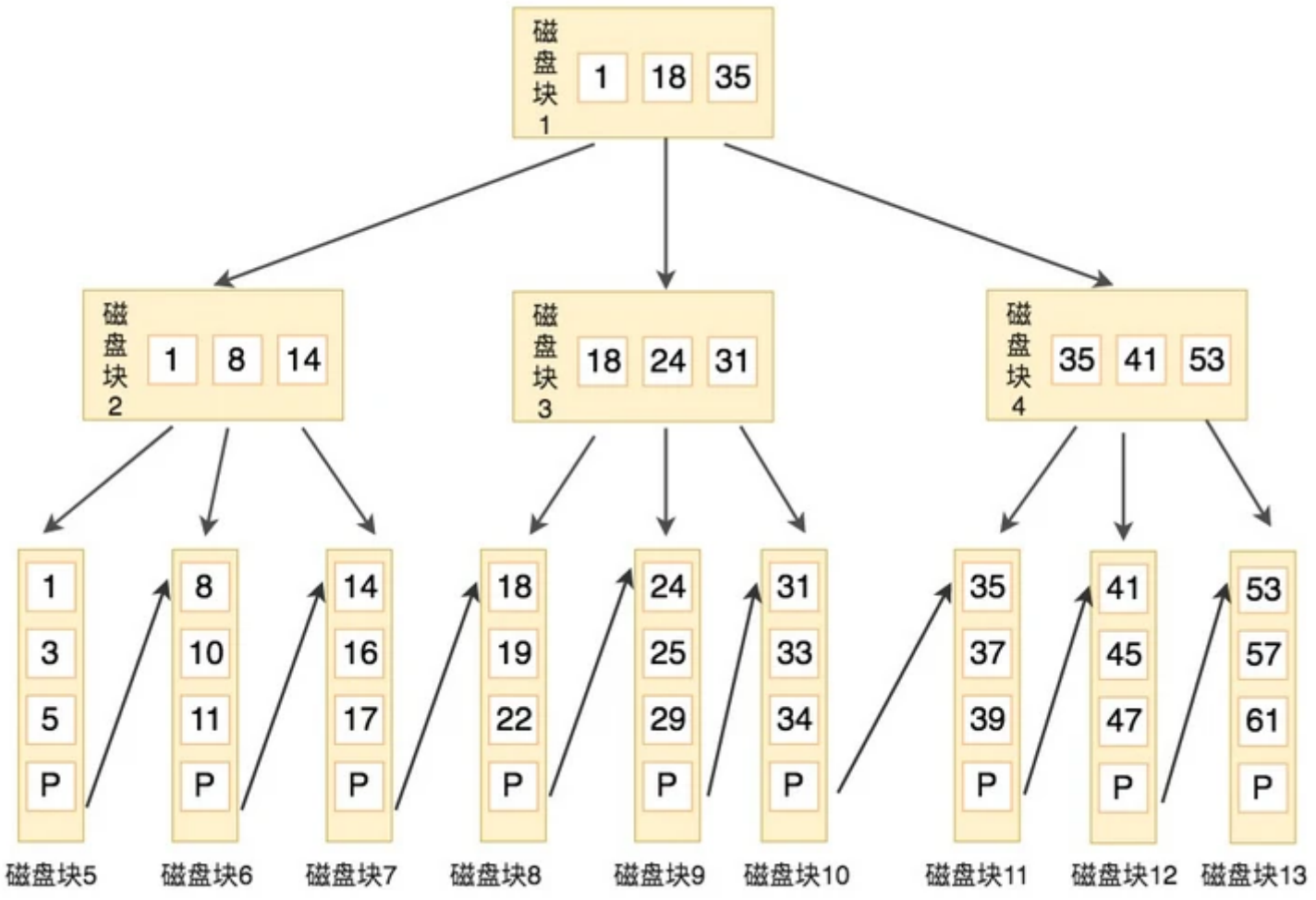
B-tree 中的每个节点根据实际情况可以包含多条数据信息和子节点，如下图所示为一个3阶的B-tree：



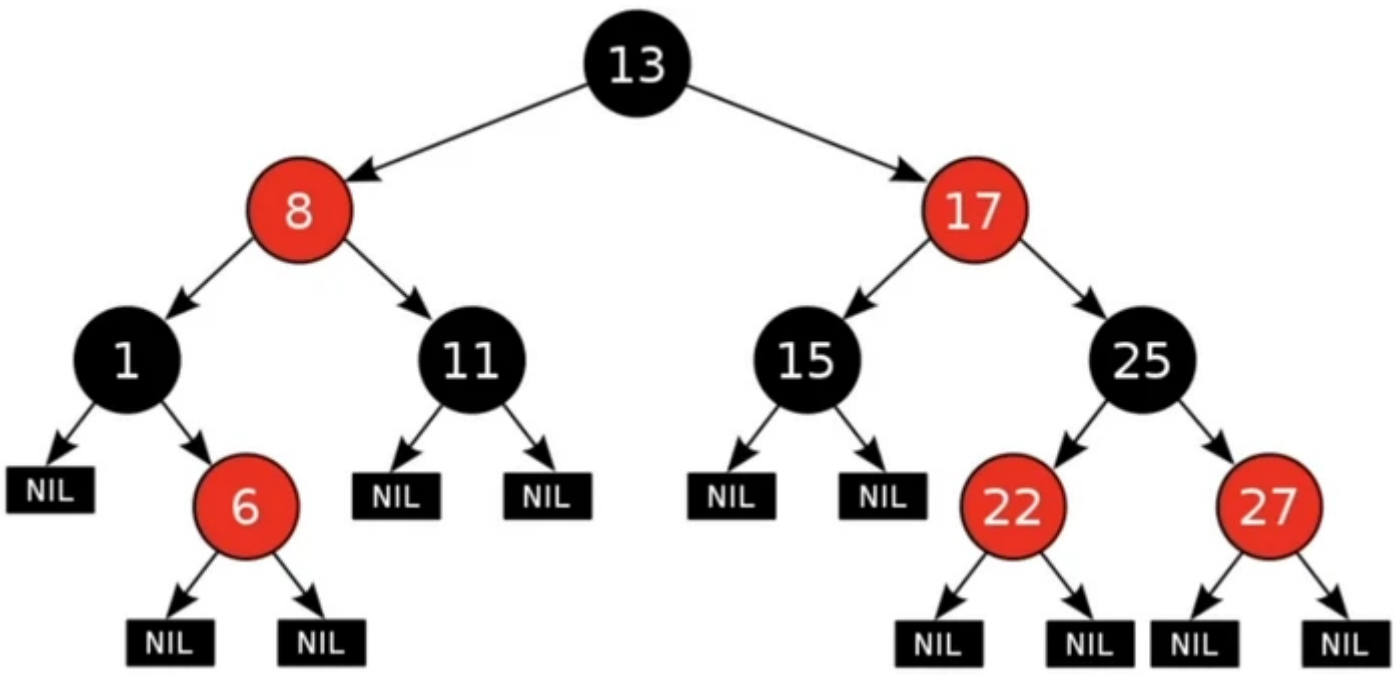
（图片来源于网络）

相对于B-tree，B+tree有以下两点不同：

- B+tree 非叶子节点只存储键值信息， 数据记录都存放在叶子节点中。而B-tree的非叶子节点也存储数据。所以B+tree单个节点的数据量更小，在相同的磁盘I/O次数下，能查询更多的节点。
- B+tree 所有叶子节点之间都采用单链表连接。适合MySQL中常见的基于范围的顺序检索场景，而B-tree无法做到这一点。



2. B+tree与红黑树的对比



（图片来源于网络）

红黑树是一种**弱平衡二叉查找树。**通过对任何一条从根到叶子的路径上各个节点着色的方式的限制，**红黑树确保没有一条路径会比其他路径长出两倍。**

对于有N个叶子结点的 **B+tree**，其搜索复杂度为 `O(logdN)` ，其中 **d**(degree) 为 **B+tree** 的度，表示节点允许的最大子节点个数为**d**个，在实际应用当中，**d**值一般是大于100的，即使数据量达到千万级别时**B+tree**的高度依然维持在3-4左右，保证了3-4次磁盘I/O操作就能查询到目标数据。

红黑树是二叉树，节点子节点个数为两个，意味着其搜索复杂度为 `O(logN)`，树的高度也会比 **B+tree** 高出不少，因此**红黑树**检索到目标数据所需经历的磁盘I/O次数更多。

3. B+tree与Hash的对比

Hash 索引结构的特殊性，其检索效率非常高，索引的检索可以一次定位，不像B-Tree 索引需要从根节点到枝节点，最后才能访问到页节点这样多次的IO访问，所以 Hash 索引的查询效率要远高于 B-Tree 索引。虽然 Hash 索引效率高，但是 Hash 索引本身由于其特殊性也带来了很多限制和弊端，主要有以下这些。

Hash 索引仅仅能满足 `=` , `IN` 和 `<=>`(表示**NULL**安全的等价) 查询，不能使用范围查询。

由于 Hash 索引比较的是进行 Hash 运算之后的 Hash值，所以它只能用于等值的过滤，不能用于基于范围的过滤，因为经过相应的Hash算法处理之后的 Hash 值的大小关系，并不能保证和Hash运算前完全一样。

Hash 索引无法适用数据的排序操作。

由于 Hash 索引中存放的是经过 Hash 计算之后的 Hash值，而且Hash值的大小关系并不一定和 Hash运算前的键值完全一样，所以数据库无法利用索引的数据来避免任何排序运算；

Hash 索引不能利用部分索引键查询。

对于组合索引，Hash 索引在计算 Hash 值的时候是组合索引键合并后再一起计算 Hash 值，而不是单独计算 Hash值，所以通过组合索引的前面一个或几个索引键进行查询的时候，Hash 索引也无法被利用。

Hash 索引依然需要回表扫描。

Hash 索引是将索引键通过 Hash 运算之后，将 Hash运算结果的 Hash值和所对应的行指针信息存放于一个 Hash 表中，由于不同索引键可能存在相同 Hash 值，所以即使取满足某个 Hash 键值的数据的记录条数，也无法从 Hash索引中直接完成查询，还是要通过访问表中的实际数据进行相应的比较，并得到相应的结果。

Hash索引遇到大量Hash值相等的情况后性能并不一定就会比B-Tree索引高。

选择性比较低的索引键，如果创建 Hash 索引，那么将会存在大量记录指针信息存于同一个Hash值相关联。这样要定位某一条记录时就会非常麻烦，会浪费多次表数据的访问，而造成整体性能低下

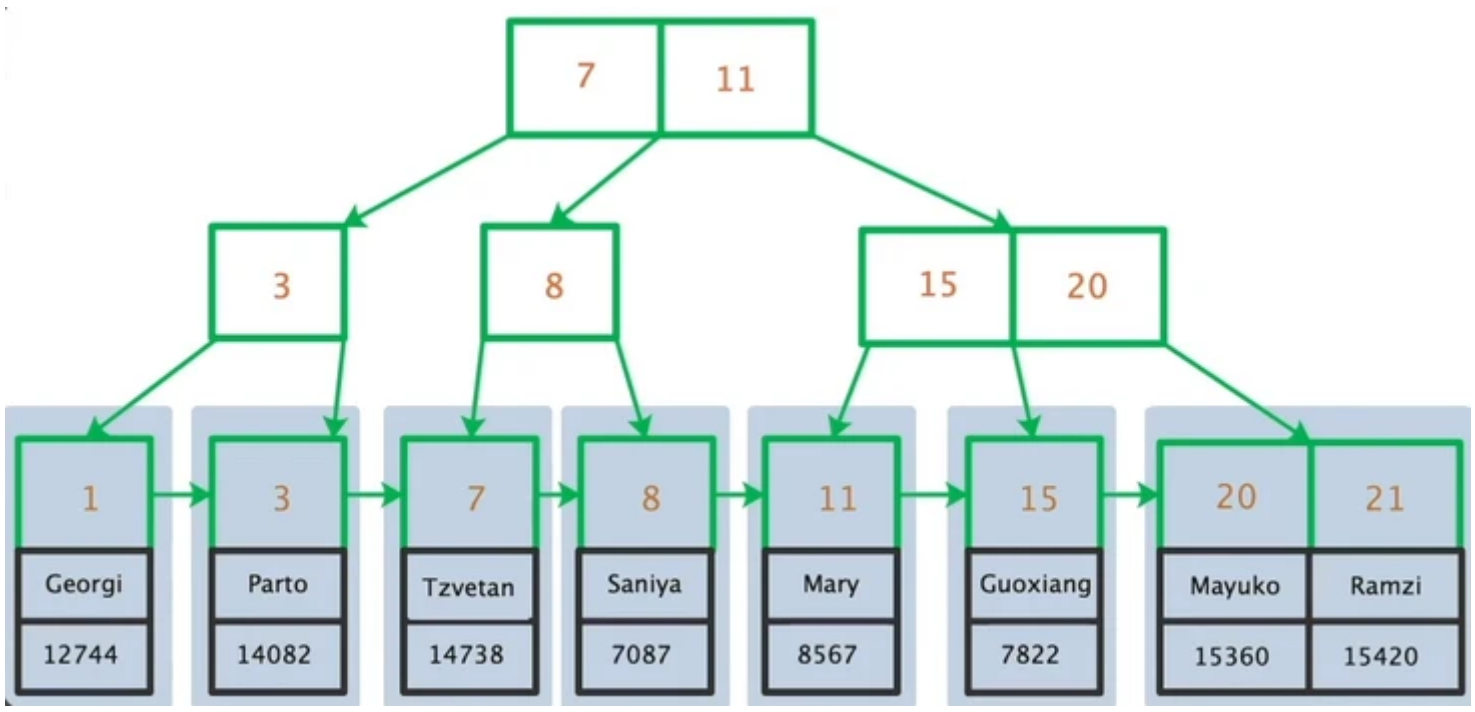
由于范围查询是MySQL数据库查询中常见的场景，Hash表不适合做范围查询，它更适合做等值查询。另外Hash表还存在Hash函数选择和Hash值冲突等问题。因此，B+tree索引要比Hash表索引有更广的适用场景。

二、按物理存储分类

MySQL索引按叶子节点存储的是否为完整表数据分为：**聚簇索引、二级索引（辅助索引）**。全表数据存储在聚簇索引中，聚簇索引以外的其他索引叫做二级索引，也叫辅助索引。

1. 聚簇索引

聚簇索引的每个叶子节点存储了一行完整的表数据，叶子节点间按id列递增连接，可以方便地进行顺序检索。

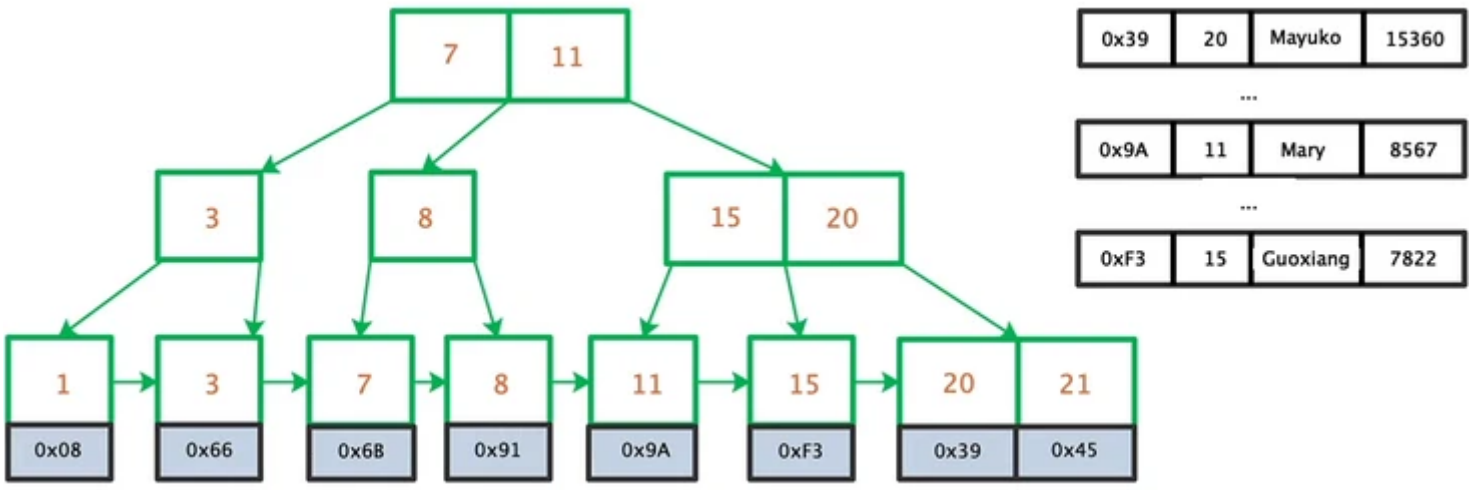


（图片来源于网络）

InnoDB表要求必须有聚簇索引，默认在主键字段上建立聚簇索引，在没有主键字段的情况下，表的第一个非空的唯一索引将被建立为聚簇索引，在前两者都没有的情况下，InnoDB将自动生成一个隐式的自增id列，并在此列上建立聚簇索引。

以MyISAM为存储引擎的表不存在聚簇索引。

MyISAM表中的主键索引和非主键索引的结构是一样的，索引的叶子节点不存储表数据，存放的是表数据的地址。所以，MyISAM表可以没有主键。

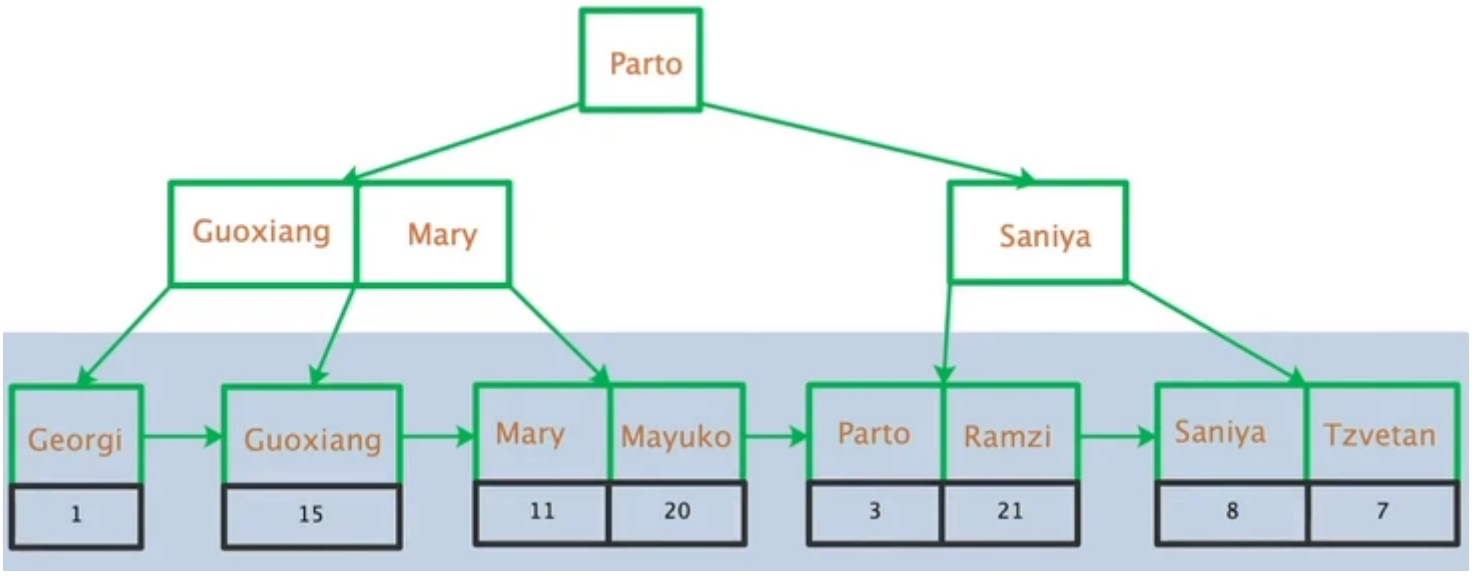


（图片来源于网络）

MyISAM表的数据和索引是分开存储的。MyISAM表的主键索引和非主键索引的区别仅在于主键索引的B+tree上的key必须符合主键的限制，非主键索引B+tree上的key只要符合相应字段的特性就可以了。

2. 二级索引

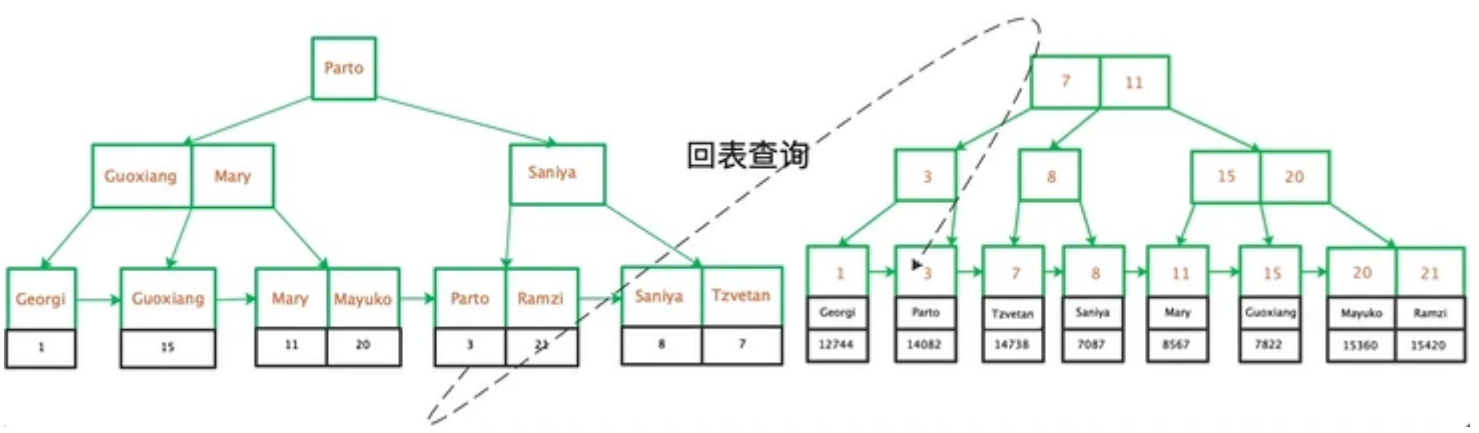
二级索引的叶子节点并不存储一行完整的表数据，而是存储了聚簇索引所在列的值。



（图片来源于网络）

回表查询

由于二级索引的叶子节点不存储完整的表数据，索引当通过二级索引查询到聚簇索引列值后，还需要回到聚簇索引也就是表数据本身进一步获取数据。



（图片来源于网络）

回表查询 需要额外的 **B+tree** 搜索过程，必然增大查询耗时。

需要注意的是，**通过二级索引查询时，回表不是必须的过程**，当**SELECT的所有字段在单个二级索引中都能够找到**时，就不需要回表，MySQL称此时的二级索引为**覆盖索引**或触发了**索引覆盖。**
可以用Explain命令查看SQL语句的执行计划，执行计划的Extra字段中若出现**Using index**，表示查询触发了**索引覆盖。**

三、按字段特性分类

MySQL索引按字段特性分类可分为：**主键索引、普通索引、前缀索引。**

1. 主键索引

建立在主键上的索引被称为**主键索引**，一张数据表只能有一个主键索引，索引列值不允许有空值，通常在创建表时一起创建。

2. 唯一索引

建立在UNIQUE字段上的索引被称为**唯一索引**，一张表可以有多个唯一索引，索引列值允许为空，列值中出现多个空值不会发生重复冲突。

3. 普通索引

建立在普通字段上的索引被称为**普通索引**。

4. 前缀索引

前缀索引是指对字符类型字段的前几个字符或对二进制类型字段的前几个bytes建立的索引，而不是在整个字段上建索引。前缀索引可以建立在类型为char、varchar、binary、varbinary的列上，可以大大减少索引占用的存储空间，也能提升索引的查询效率。

四、按索引字段个数分类

MySQL索引按字段个数分类可分为：**单列索引、联合索引（复合索引、组合索引）**。

1. 单列索引

建立在单个列上的索引被称为单列索引。

2. 联合索引（复合索引、组合索引）

建立在多个列上的索引被称为联合索引，又叫复合索引、组合索引。