**索引(普通索引、唯一索引、全文索引、索引匹配原则、索引命中等)**

**优点：**

第一，可以大大加快 数据的检索速度，这也是创建索引的最主要的原因。

第二，在使用分组和排序 子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

第三，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

**缺点：**

第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立联合索引，那么需要的空间就会更大。

第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

**1.B+树索引**

就是传统意义上的索引，最为常用最为有效的索引。在数据库中，B+树的高度一般都是2-4层，一般机械硬盘每秒可做100次IO，2-4次IO意味着查询时间只需要0.02-0.04；

分为聚集索引和辅助索引，聚集索引对范围性检索的速度快，而辅助索引对单行的检索速度快。

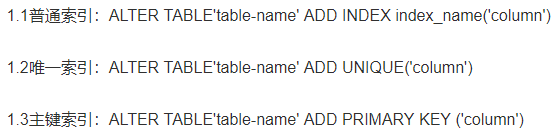
这两大索引类型下，还可分为四个小类。讲完小类之后，再继续讲聚集和辅助索引

1，普通索引：最基本的索引，没有任何限制，是我们大多数情况下使用到的索引。

2，唯一索引：与普通索引类型，不同的是唯一索引的列值必须唯一，但允许为空值，主键索引是特殊的唯一索引，不允许有空值。

3，全文索引：全文索引（FULLTEXT）仅可以适用于MyISAM引擎的数据表；作用于CHAR、VARCHAR、TEXT数据类型的列。

4，组合索引：将几个列作为一条索引进行检索，使用最左匹配原则。



**建立索引的原则：**

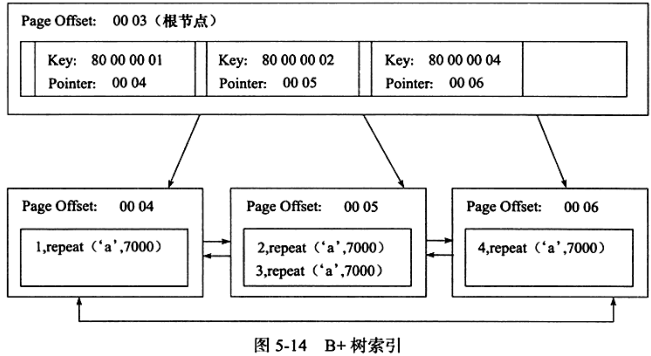
**最左前缀匹配原则**，联合索引中（a,b,c），索引的建立是按照a的顺序优先的，即能保证a有序，其他不一定，但是如果a是相同的，那么相同的a里面，b是有序的，但是c不一定，如果a，b都相同了，那么c是有序的。

MySQL会一直向右匹配直到遇到范围查询（>,<,BETWEEN,LIKE）就停止匹配，比如： a = 1 AND b = 2 AND c > 3 AND d = 4，如果建立 （a,b,c,d）顺序的索引，d是用不到索引的，如果建立（a,b,d,c）的索引，则都可以用到，a,b,d的顺序可以任意调整。至于任意调整不是不符合最左匹配，是因为查询优化器会判断纠正这条sql语句该以什么样的顺序执行效率最高，最后才生成真正的执行计划。但是如果没有出现第一列索引，那么不能根据第一列查找数据，就会出现不使用索引的情况。

数据页的某些参数：file-type-index = 0x45BF表明叶节点，page-level为0x0000表明数据页，即叶子节点page-level为0001表明根节点。

聚集索引：按照每张表的主键构造一个b+树，并且叶节点中存放着整张表的行记录数据，所以也让聚集索引的叶节点成为数据页，每个数据页都通过一个双向链表来进行链接。实际的数据页只能按照一棵B+树进行排序，因此每张表只能拥有一个聚集索引。

许多情况下，查询优化器偏向于使用聚集索引，由于聚集索引定义了逻辑顺序，能够特别快的针对范围值进行查询



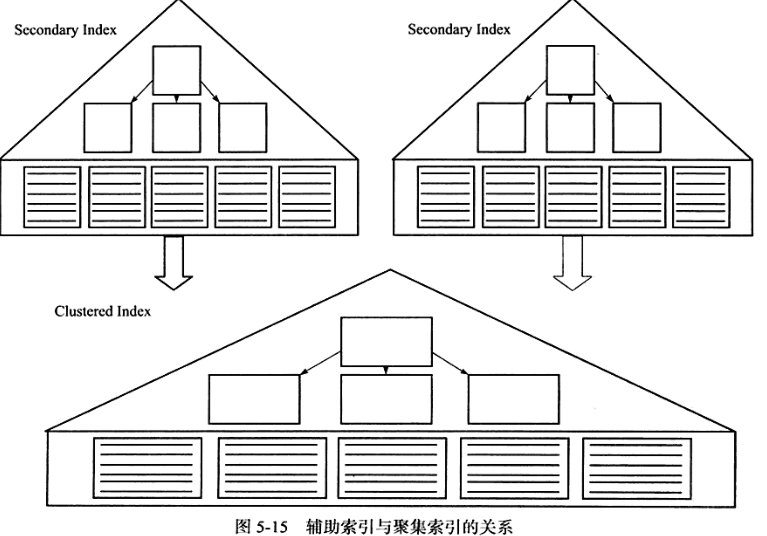
非数据节点即非叶子节点，存放的仅仅只是键值和指向数据页偏移量的指针，不是完整的行记录，叶子节点上保存着完整的行记录。

**聚集索引**是按照逻辑顺序排的，不然还得按照特定顺序存储物理记录，维护成本会很高。

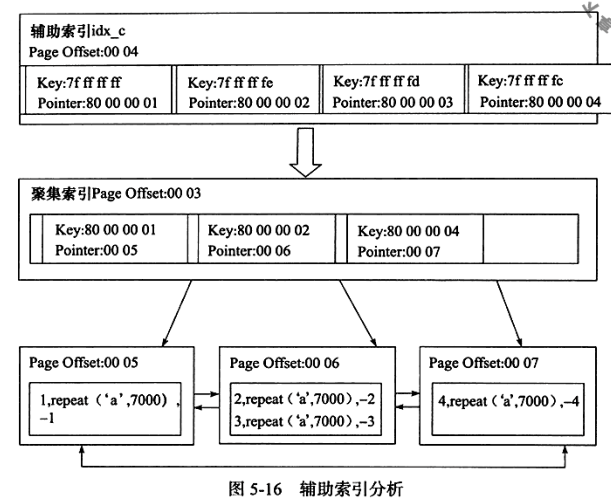
两点：页通过双向链表连接，页按照主键顺序排列，另一点，每个页中的记录，即行数据也是通过双向链表进行维护，物理存储上可以不按主键存储，因为不是按照物理顺序来的。

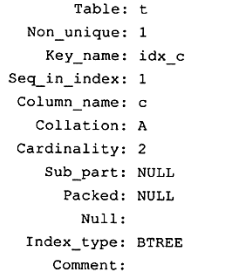
对主键的排序查找和范围查找速度飞快，叶节点的数据就是我们想要的数据

辅助索引：叶节点不包含行的全部数据，叶节点除了包含键值以外，每个也级别中的索引行还包含了一个书签，书签用来告诉存储引擎，哪里可以找到与**索引对应的行数据**。辅助索引的书签就是相应行数据的聚集索引键。



**辅助索引**的存在不影响数据在聚集索引中的组织，因此每张表上可以有多个辅助索引。辅助索引如果要查找一个完整记录，需要两个步骤，先从辅助索引中找到叶子节点指向聚集索引的书签，然后通过聚集索引，找到行记录，一次查找算一次逻辑io。





Non-unique：非唯一索引，值为1，唯一索引，值为2

Key-name：索引名字

Seq-in-index：索引中该列的位置

Column-name：索引列的名称

Collation：索引方式，B+树为A，heap为Null

Cardinality:索引中谓一致的数目估计值。优化器会根据这个值判断是否使用这个索引

Sub-part表示是否是列的部分被索引

**Explain 参数解析**

Id：查询序列号，值越大优先级越高，越先执行，如果id相同，按上到下顺序

Select-type：查询类型，simple简单查询，union联合查询，subquery子查询等

Type：显示连接使用的类型，从最优到最差

--system：表仅有一行，系统表，const特例

--const：用于主键与常数值的比较，结果只有一条数据

--ref：查询条件列使用了索引列，并且不是主键也不唯一，得到多条数据，在小范围中扫描

--ref-eq：使用了主键或者唯一性索引进行查找

--range：有范围的索引扫描，包括between 大于小于，in 和or也是

--index：①只查询索引列时出现，（索引是已经排序好的，实际数据没有排序好）

--------②order by索引列，此时不需要再次排序

--all：全表扫描

Ref：显示索引的哪一列被使用了

Extra：using where：光靠索引定位不了，还得使用where判断

Using index：用到了索引覆盖，效率高

Using temporary：常用于orderby和groupby 结果排序时使用临时表

Using filesort：索引得到的数据不是按照需要排序的，需要进行另外的排序

Using intersect：使用两个索引得到结果求交集的数学运算

**什么时候使用索引：**

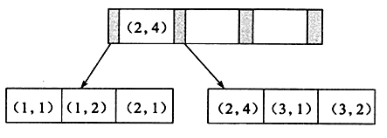
低选择性：取值范围很小，如性别，地区等

高选择性：取值范围广，基本没有重复，

访问高选择性字段并从表中取出很少一部分行时，对这个字段添加b+索引很有必要，但是如果虽然字段是高选择的，但是数据量很大，就不会使用索引了，一般取出的量超过20%就不会使用索引了，而是进行全表查询。

**联合索引**

对表上多个列进行索引



先**按1进行排序**，在1**相同**的情况下，**按照2进行排序**。

如果一张表buy\_log(userid , buy\_date);使用了普通索引userid和联合索引(userid , buy\_date),

当使用select \* from buy\_log where userid = 2;



**优化器会选择userid索引，**因为该索引的叶子节点包含的单个键值，理论上一页能够存放更多的记录

但是在select \* from buy\_log where userid = 1 order by buy\_date desc limit 3;



使用联合索引，因为在联合索引中，id为1的按照日期已经进行降序排序了，不需要在对其进行额外的排序操作，若强制使用userid，extra值会出现using filesort，表明还需要一次额外的排序，因为userid中的buy\_date是未排序的。

**覆盖索引**

从辅助索引中就可以得到查询的记录，而不需要查询聚集索引中的记录，使用覆盖索引的好处是辅助索引不包含整个行记录所有信息，因此比聚集索引占的空间小，就可以容纳更多的信息，相同数目的节点，会有更低的树的高度，从而减少查找节点逻辑IO操作

不能覆盖索引的情况：如果用户需要选择数据是整行信息，此时辅助索引不能覆盖聚集索引，因为在辅助索引找到指定数据后，还需要一次书签访问来查找整行数据信息。虽然在辅助索引中是有序的，但是在通过书签找数据的时候就会变成无序的了，因此变为磁盘随机IO操作，在数据量大的时候会效率低，一般在访问数据占总数据的20%以下使用辅助索引，以上使用聚集索引

可以使用use index 来提示优化器选择什么，但是优化器最终的选择还是根据自己进行判断。

**Mysql的Multi-Range Read优化**

减少磁盘的随机访问，并且将随机访问转化为较为顺序的数据访问，这对sql查询语句中的IO操作可带来性能的极大提升

优点：

①使数据变得较为有序，在查询辅助索引时，首先根据得到的结果，按照**主键进行排序**，并按照主键排序的顺序进行书签查找。因为辅助索引最终是通过聚集索引得到行数据的，而聚集索引就是按照主键进行排序的，如果将辅助索引的结果按照主键排序，那么只需要根据双向链表，就能最大程度减少查找的过程。

②减少缓冲池中页被替换的次数。

在innoDB和MyISAM中，MRR的工作方式：

①将查询得到的辅助索引键值存放于一个缓存中，这是缓存中的数据是根据辅助索引键值排序的

②将缓存的键值根据rowID进行排序

③根据rowId的排序来访问实际的数据文件。

Mysql的ICP（index condition pushdown）优化

之前mysql数据库进行索引查询时，先根据索引查找记录，然后根据where条件来过滤记录。有了ICP之后，会在取出索引的同时，判断是否可以进行where条件过滤，即将where的部分富哦率操作放在了存储引擎。

能进行ICP的条件是，该索引可以覆盖到的范围，即索引中需要包含where后面的某些条件