**数据库索引（索引类型+自适应索引+B树B+树索引）**

**数据库索引**，是数据库管理系统中**一个排序的数据结构**以**协助快速查询、更新数据库表中数据**。索引的实现通常使用**B树**及其变种**B+树**。

**索引的目的**在于**提高查询效率**，索引原理是通过不断的缩小想要获得数据的范围来筛选出最终想要的结果。每次查找数据时**把磁盘IO次数控制在一个很小的数量级**，最好是常数数量级。

mysql的**Innodb存储引擎**中的索引大概分三种，**哈希索引**，**全文索引**，和常用的**B+树索引。**

**InnoDB**给MySQL的表提供了**事务处理**、**回滚**、**崩溃修复能力**和**多版本并发控制的事务安全**。在MySQL从3.23.34a开始包含InnnoDB。它是MySQL上第一个提供**外键约束**的表引擎。而且InnoDB对事务处理的能力，也是其他存储引擎不能比拟的。靠后版本的MySQL的默认存储引擎就是InnoDB。

InnoDB的优势在于提供了**良好的事务处理**、**崩溃修复能力**和**并发控制**。缺点是**读写效率较差**，**占用的数据空间相对较大**。

**自适应哈希索引**

InnoDB存储引擎会**监控**对**表上索引的查找**，如果观察到建立哈希索引可以带来速度的提升，则建立哈希索引，所以称之为自适应（adaptive）的。自适应哈希索引通过**缓冲池的B+树**构造而来，因此建立的速度很快。而且不需要将整个表都建哈希索引，InnoDB存储引擎会自动根据**访问的频率和模式**来为某些页建立哈希索引。常用于连接（join）操作。

哈希索引是系统自己管理维护的，且对范围查询无效，生成的条件相对比较苛刻，**当用户多次调用相同的查询sql**，例如select \* from user where user\_name = xxx，这样的sql连续调用多次（貌似要超过100次），此时mysql实例会自生成哈希索引，且建立索引的速度很快，大大的加快的查询的效率，mysql**默认是打开**自适应的哈希索引的。

**B+树索引**

B+树索引就是传统意义上的索引，是关系型数据库中最常用最有效的索引。B+树是从最早的平衡二叉树演变而来，但是B+树不是一个二叉树。B+中的B不代表二叉(Binary),而是代表平衡(Balance)。

【**注意**】B+树索引**并不能找到一个键值对应的具体行**。b+树索引只能查到被查找数据行**所在的页**，然后数据库通过把页读入内存，再在内存中查找，最后得到结果。

数据库中B+树索引分为聚集索引（clustered index）和非聚集索引（secondary index）.这两种索引的共同点是内部都是B+树，高度都是平衡的，叶节点存放着所有数据。不同点是叶节点是否存放着一整行数据。

(1) 聚集索引

Innodb存储引擎表是索引组织表，即表中数据按主键顺序存放。而聚集索引就是按每张表的主键构造一颗B+树。并且叶节点存放整张表的行记录数据。每张表只能有一个聚集索引（一个主键）。聚集索引的另一个好处是它对于主键的排序查找和范围的速度非常快。叶节点的数据就是我们要找的数据。

(2) 辅助索引

辅助索引（也称非聚集索引）。叶级别不包含行的全部数据，叶级别除了包含行的键值以外，每个索引行还包含了一个书签（bookmark），该书签告诉innodb存储引擎，哪里可以找到与索引对应的数据。

辅助索引的存在并不影响数据再聚集索引中的组织，因此一个表可以有多个辅助索引。当通过辅助索引查找数据时，innodb会遍历辅助索引并通过叶级别的指针获得指向主键索引的主键。然后再通过主键索引找到一行完整的数据。

一般使用磁盘I/O次数评价索引结构的优劣。先从B-Tree分析，根据B-Tree的定义，可知**检索一次最多需要访问h个节点**。数据库系统的设计者巧妙利用了**磁盘预读**原理，将**一个节点的大小设为等于一个页**，这样每个节点只需要一次I/O就可以完全载入。为了达到这个目的，在实际实现B-Tree还需要使用如下技巧：每次**新建节点时，直接申请一个页的空间**，这样就保证一个节点物理上也存储在一个页里，加之计算机存储分配都是按页对齐的，就实现了一个node只需一次I/O。

**1）B-树**

B-Tree是一种多路搜索树（并不是二叉的）：

1.定义任意非叶子结点最多只有M个儿子；且M>2；

2.根结点的儿子数为[2, M]；

3.除根结点以外的非叶子结点的儿子数为[M/2, M]；

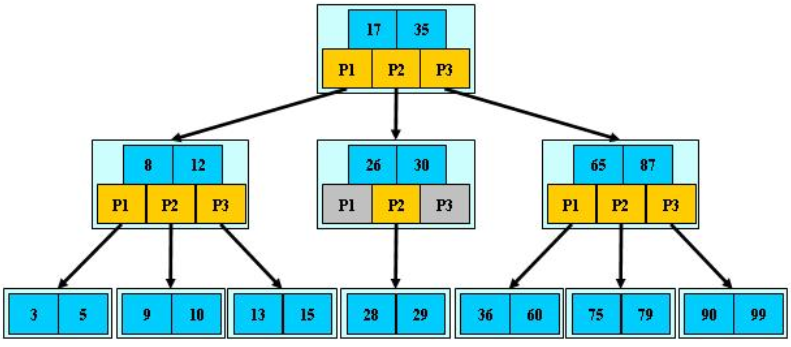
4.每个结点存放至少M/2-1（取上整）和至多M-1个关键字；（至少2个关键字）

5.非叶子结点的关键字个数=指向儿子的指针个数-1；

6.非叶子结点的关键字：K[1], K[2], …, K[M-1]；且K[i] < K[i+1]；

7.非叶子结点的指针：P[1], P[2], …, P[M]；其中P[1]指向关键字小于K[1]的子树，P[M]指向关键字大于K[M-1]的子树，其它P[i]指向关键字属于(K[i-1], K[i])的子树；

8.所有叶子结点位于同一层。



**B-树的特性：**

1.关键字集合分布在整颗树中；

2.任何一个关键字出现且只出现在一个结点中；

3.搜索有可能在非叶子结点结束；

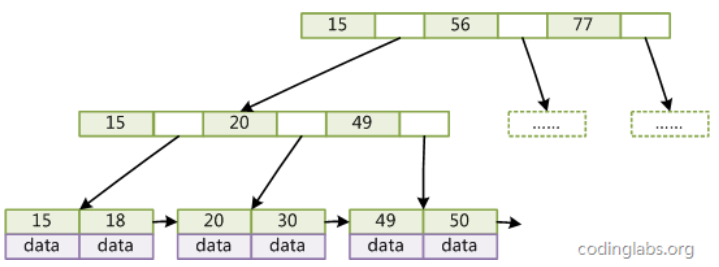
4.其搜索性能等价于在关键字全集内做一次二分查找；

5.自动层次控制；

**B-树的搜索，从根结点开始，对结点内的关键字（有序）序列进行二分查找，如果命中则结束，否则进入查询关键字所属范围的儿子结点；重复，直到所对应的儿子指针为空，或已经是叶子结点。**

**2）B+树**

　　B+树非叶节点中存放的关键码并不指示数据对象的地址指针，**非叶节点只是索引部分**。所有的**叶节点在同一层上，包含了全部关键码和相应数据对象的存放地址指针**，且叶节点按关键码从小到大顺序链接。如果实际数据对象按加入的顺序存储而不是按关键码次数存储的话，叶节点的索引必须是稠密索引，若实际数据存储按关键码次序存放的话，叶节点索引时稀疏索引。



**注：所有的key都会在叶子结点中。（mysql中使用的是B+树作为索引）**

**B+树的特性：**

1.所有关键字都出现在叶子结点的链表中（稠密索引），且链表中的关键字恰好是有序的；

2.不可能在非叶子结点命中；

3.非叶子结点相当于是叶子结点的索引（稀疏索引），叶子结点相当于是存储（关键字）数据的数据层；

4.更适合文件索引系统。

在B+Tree的每个叶子节点增加一个指向相邻叶子节点的指针，就形成了**带有顺序访问指针的B+Tree**。做这个优化的目的是为了**提高区间访问的性能**。

　　B+树有2个头指针，一个是树的根节点，一个是最小关键码的叶节点。所以 B+树有两种搜索方法：

**一种是按叶节点自己拉起的链表顺序搜索。**

**一种是从根节点开始搜索，和B树类似，不过如果非叶节点的关键码等于给定值，搜索并不停止，而是继续沿右指针，一直查到叶节点上的关键码。所以无论搜索是否成功，都将走完树的所有层。**

B+ 树中，数据对象的插入和删除仅在叶节点上进行。

**B-树与B+树这两种处理索引的数据结构的不同之处：**

　　1、B-树中同一键值不会出现多次，并且它有可能出现在叶结点，也有可能出现在非叶结点中。而B+树的键一定会出现在叶结点中，并且有可能在非叶结点中也有可能重复出现，以维持B+树的平衡。

　　2、因为B-树键位置不定，且在整个树结构中只出现一次，虽然可以节省存储空间，但使得在插入、删除操作复杂度明显增加。B+树相比来说是一种较好的折中。

　　3、B-树的查询效率与键在树中的位置有关，最大时间复杂度与B+树相同(在叶结点的时候)，最小时间复杂度为1(在根结点的时候)。而B+树的时候复杂度对某建成的树是固定的。

**选用B+、B-树原因：**

索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储的磁盘上。这样的话，索引查找过程中就要产生磁盘I/O消耗，相对于内存存取，I/O存取的消耗要高几个数量级，所以评价一个数据结构作为索引的优劣最重要的指标就是在查找过程中磁盘I/O操作次数的渐进复杂度。换句话说，索引的结构组织要尽量减少查找过程中磁盘I/O的存取次数。

**索引的优点：**

第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

第二，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。

第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

第四，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。

第五，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。

**索引的缺点**

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，如果要建立聚簇索引，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。

**应该在这些列上创建索引：**

1、在经常需要搜索的列上，可以加快搜索的速度；

2、在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；

3、在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；

4、在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；

5、在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；

6、在经常使用在WHERE子句中的列上面创建索引，加快条件的判断速度。

**不应该创建索引的的这些列具有下列特点：**

1、对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。

2、对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。

3、对于那些定义为text, image和bit数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少。

4、当修改性能远远大于检索性能时，不应该创建索引。这是因为，修改性能和检索性能是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。