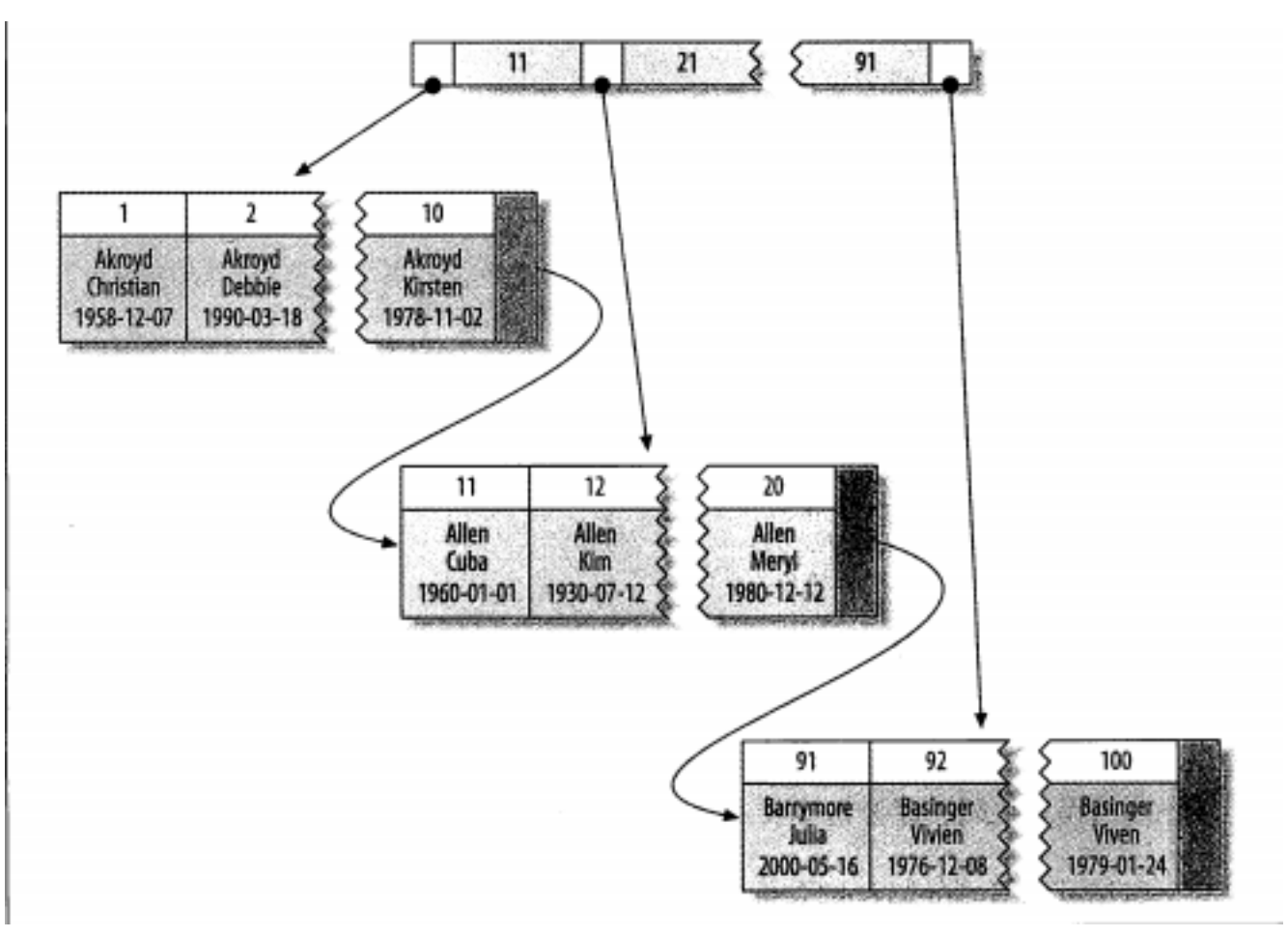
详解聚簇索引

一、聚族索引的构造

聚簇索引并不是一种单独的索引类型，而是一种数据存储方式。具体的细节依赖于其实现方式，但InnoDB的聚族索引实际上在同一个结构中保存了B-Tree索引和数据行。当表有聚族索引时，它的数据行存放在索引的叶子页中。术语“聚族”表示数据行和相邻的键值紧凑的存储在一起。因为无法同时把数据行放在两个不同的地方，所以一个表只能有一个聚族索引。

因为是存储引擎负责实现索引，因此不是所有的存储引擎都支持聚族索引。这里我们主要关注InnoDB，但是这里讨论的原理对于任何支持聚族索引的存储引擎都是适用的。

下面展示了聚族索引中的记录是如何存放的。注意到，叶子页包含了行的全部数据，但是节点页只包含了索引列。



在InnoDB中通过主键聚集数据，这也就是说上图中“被索引的列”就是主键列。如果没有定义主键，InnoDB会选择一个唯一的非空索引代替。如果没有这样的索引，InnoDB会隐式定义一个主键来作为聚族索引。InnoDB只聚集在同一个页面中的记录。包含相邻键的页面可能会相距甚远。

聚族主键可能对性能有帮助，但也可能导致严重的性能问题。所以需要仔

细的考虑聚族索引，尤其是将表的引擎从InnoDB改成其他引擎的时候。

二、聚族索引的优点

- 可以把相关数据保存在一起。例如实现电子邮件时，可以根据用户ID来聚集数据，这样只需要从磁盘读取少数的数据页就能获取某个用户的全部邮件。如果没有使用聚族索引，则每封邮件都可能导致一次磁盘I/O；
- 数据访问更快。聚族索引将索引和数据保存在同一个B-Tree中，因此从聚族索引中获取数据通常比在非聚族索引中查找更快。
- 使用覆盖索引扫描的查询可以直接使用节点中的主键值。

三、聚族索引的缺点

- 聚簇数据最大限度的提高了I/O密集型应用的性能，但如果数据全部都放在内存中，则访问的顺序就没有那么重要了，聚簇索引也就没有那么优势了；
- 插入速度严重依赖于插入顺序。按照主键的顺序插入是加载数据到InnoDB表中速度最快的方式。但如果不是按照主键顺序加载数据，那么在加载完成后最好使用OPTIMIZE TABLE命令重新组织一下表。
- 更新聚簇索引列的代价很高，因为会强制InnoDB将每个被更新的行移动到新的位置。
- 基于聚簇索引的表在插入新行，或者主键被更新导致需要移动行的时候，可能面临“页分裂”的问题。当行的主键值要求必须将这一行插入到某个已满的页中时，存储引擎会将该页分裂成两个页面来容纳该行，这就是一次分裂操作。页分裂会导致表占用更多的磁盘空间。
- 聚簇索引可能导致全表扫描变慢，尤其是行比较稀疏，或者由于页分裂导致数据存储不连续的时候。
- 二级索引（非聚簇索引）可能比想象的要更大，因为在二级索引的叶子节点包含了引用行的主键列。
- 二级索引访问需要两次索引查找，而不是一次。

备注：有关二级索引需要两次索引查找的问题？答案在于二级索引中保存的“行指针”的实质。要记住，二级索引叶子节点保存的不是指向行的物理位置的指针，而是行的主键值。这意味着通过二级索引查找行，存储引擎需要找到二级索引的叶子节点获得对应的主键值，然后根据这个值去聚簇索引中

查找到对应的行。这里做了重复的工作：两次B-Tree查找而不是一次。对于InnoDB，自适应哈希索引能够减少这样的重复工作。

四、InnoDB和MyISAM的数据分布对比

聚簇索引和非聚簇索引的数据分布有区别，以及对应的主键索引和二级索引的数据分布也有区别。

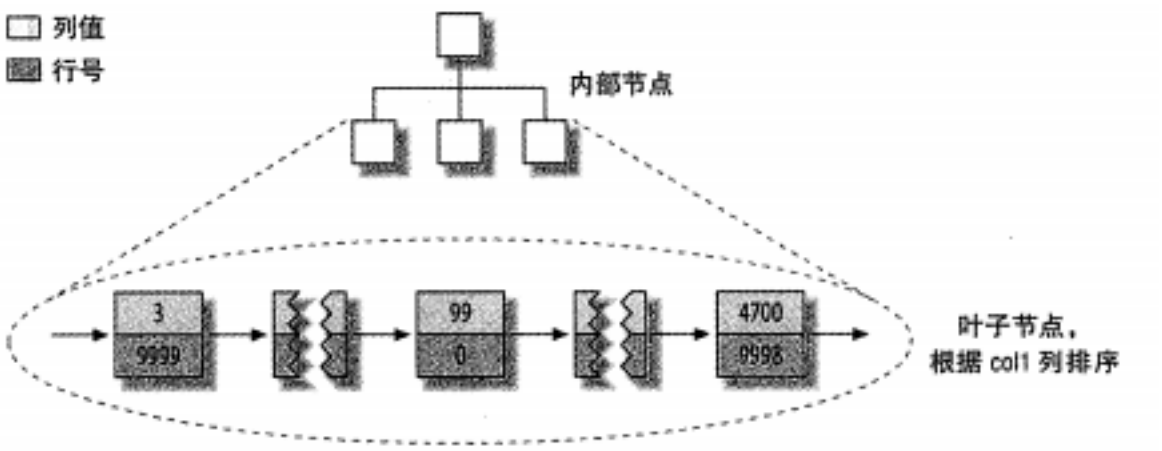
1、MyISAM的主键索引和二级索引

MyISAM的数据分布非常简单，MyISAM按照数据插入的顺序存储在磁盘上。在行的旁边显示了行号，从0开始递增。因为行是定长的，所以MyISAM可以从表的开头跳过所需的字节找到需要的行。这种分布方式很容易创建索引。并且，MyISAM中主键索引和其他索引在结构上没有什么不同。主键索引就是一个名为primary的唯一非空索引。如下图：

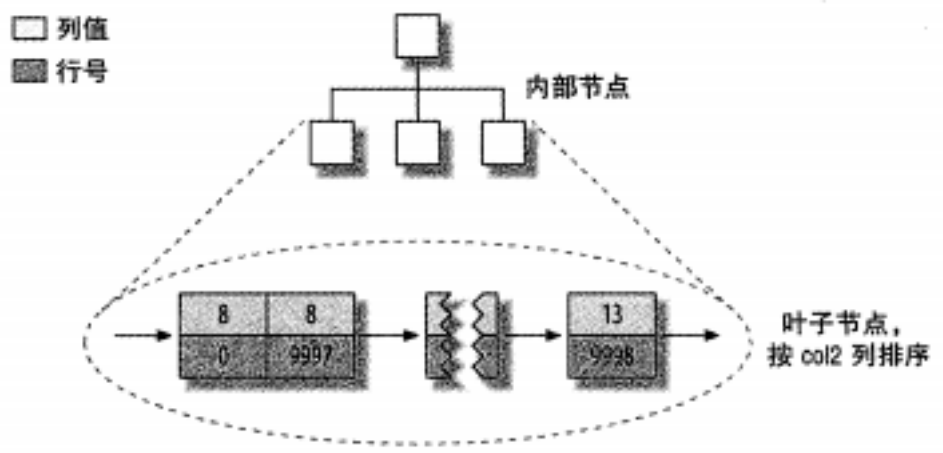
1、MyISAM数据行分布

行号	col1	col2
0	99	8
1	12	56
2	3000	62
9997	18	8
9998	4700	13
9999	3	93

2、MyISAM的主键分布

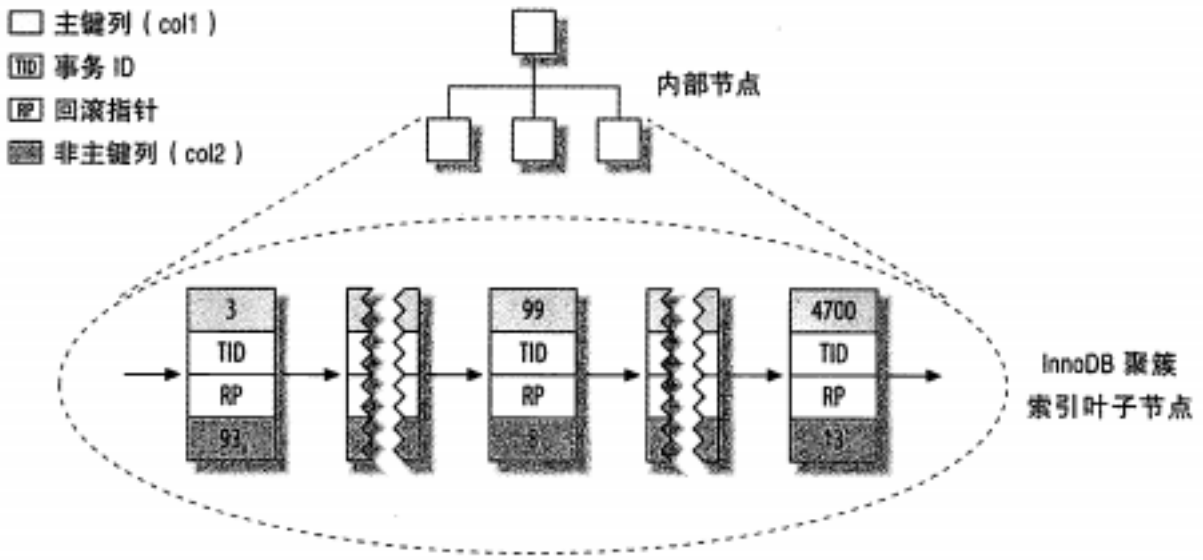


3、MyISAM上的其他索引分布



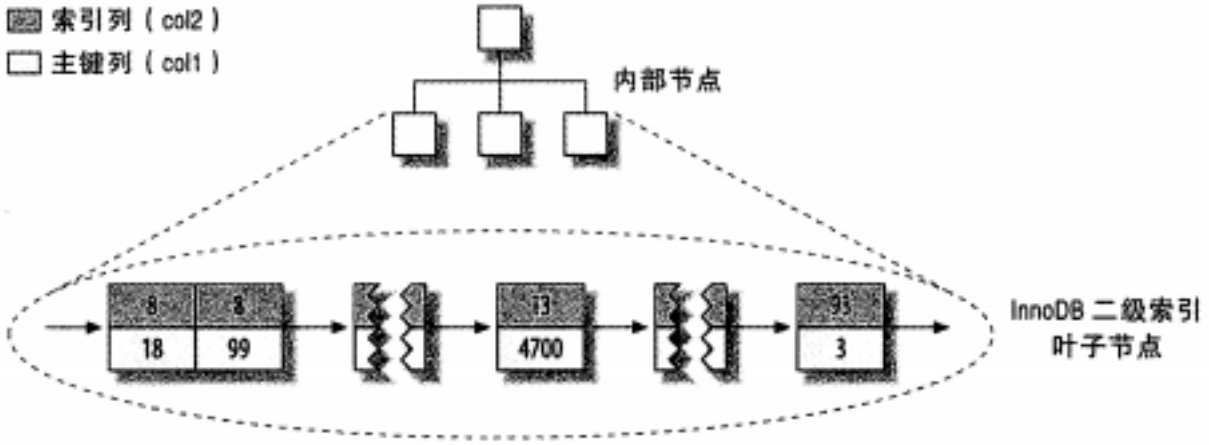
2、InnoDB的主键索引和二级索引

InnoDB的数据分布，因为InnoDB支持聚簇索引，索引使用非常不同的方式存储这样的数据，如下图：

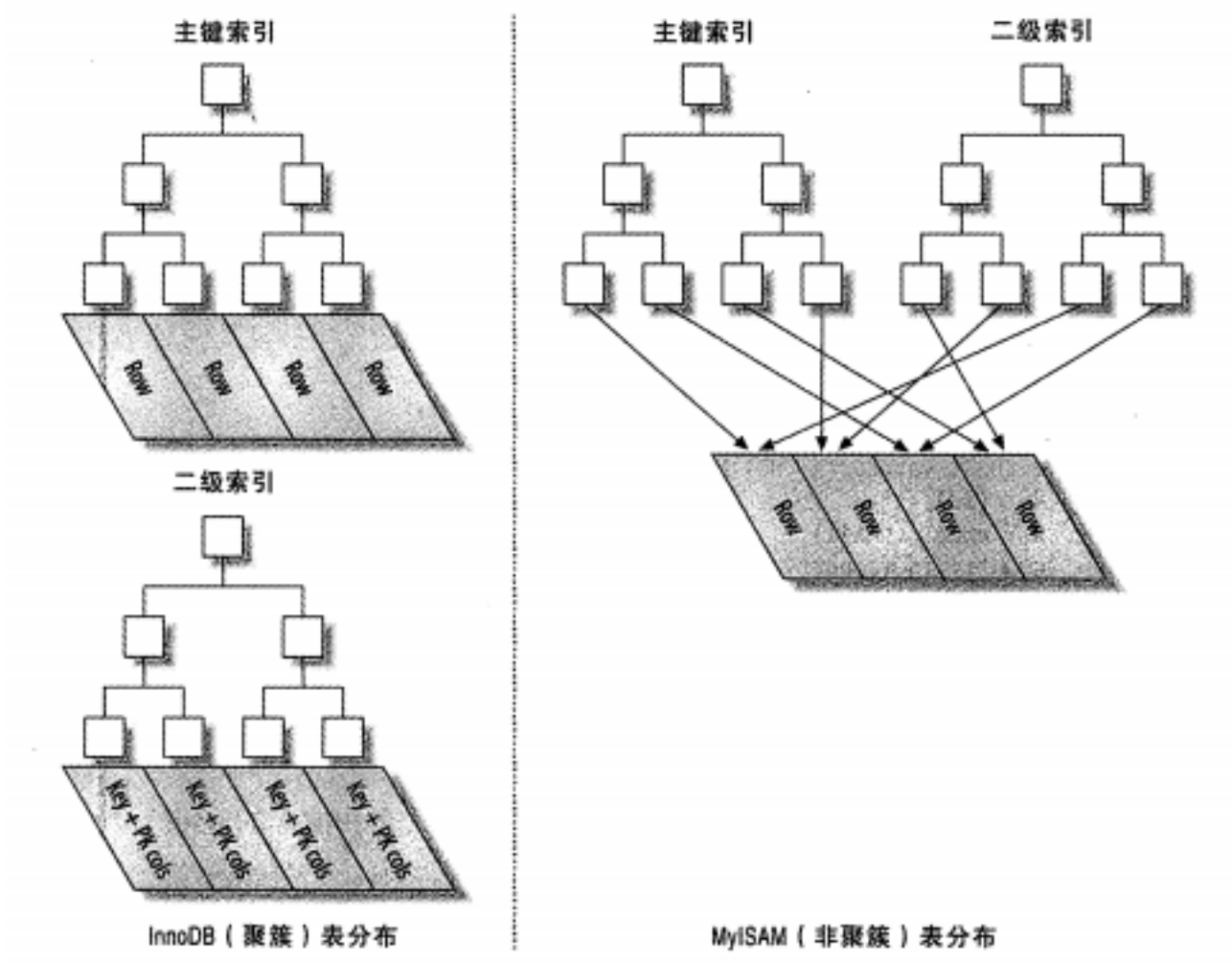


仔细查看，会注意到该图显示了整个表，而不是只有索引。因为在InnoDB中，聚簇索引“就是”表，所以不像MyISAM那样需要独立的行存储。聚簇索引的每个叶子节点都包含了主键值、事务ID、用于事务和MVCC的回滚指针以及所有的剩余列。如果主键是一个列前缀索引，InnoDB也会包含完整的主键列和剩下的其他列。

还有一点和MyISAM的不同是，InnoDB的二级索引和聚簇索引很不相同。InnoDB二级索引的叶子节点中存储的不是“行指针”，而是主键值，并以此作为指向行的“指针”。这样的策略减少了当出现航移动或者数据页分裂时二级索引的维护工作。使用主键值当作指针会让二级索引占用更多的空间，换来的好处是，InnoDB在移动行时无需更新二级索引中的这个“指针”。下图就是InnoDB的二级索引：



3、MyISAM和InnoDB的对比



五、在InnoDB表中按主键顺序插入行

如果正在使用InnoDB表并且没有什么数据需要聚集，那么可以定义一个代理键作为主键，这种主键的数据应该和应用无关，最简单的方法是使用auto_increment自增列。这样可以保证数据行是按照顺序写入，对于根据主键做关联操作的性能也会更好。

最好避免随机的聚簇索引，特别对于I/O密集型的应用。例如，从性能的角度考虑，使用UUID作为聚簇索引会很糟糕：它使得聚簇索引的插入变得完全随机，这是最坏的情况，使得数据没有任何聚集特性。通过测试，向UUID主键插入行不仅花费的时间更长，而且索引占用的空间也更大。这一方面是由于主键字段更长，另一方面毫无疑问是由于页分裂和碎片导致的。

这是由于当主键的值是顺序的，则InnoDB把每一条记录都存储在上一条

记录的后面。当达到页的最大填充因子时（InnoDB默认的最大填充因子是页大小的15/16，留出的部分空间用于以后修改），下一条记录就会写入新的页中。一旦数据按照这样顺序的方式加载，主键页就会近似于被顺序的记录填满，这也是所期望的结果。

而当采用UUID的聚簇索引的表插入数据，因为新行的主键值不一定比之前的插入值大，所以InnoDB无法简单的总是把新行插入到索引的最后，而是需要为新的行寻找合适的位置----通常是已有数据的中间位置----并且分配空间。这会增加很多额外的工作，并导致数据分布不够优化。下面是总结的一些缺点：

- 写入目标页可能已经刷到磁盘上并从缓存中移除，或者是还没有被加载到缓存中，InnoDB在插入之前不得不先找到并从磁盘读取目标页到内存中，这将导致大量的随机I/O；
- 因为写入是乱序的，InnoDB不得不频繁的做页分裂操作，以便为新的行分配空间。页分裂会导致移动大量数据，一次插入最少需要修改三个页而不是一个页。
- 由于频繁的页分裂，页会变得稀疏并被不规则的填充，所以最终数据会有碎片。
- 把这些随机值载入到聚簇索引以后，需要做一次optimize table来重建表并优化页的填充。

注意：顺序主键也有缺点：对于高并发工作负载，在InnoDB中按主键顺序插入可能会造成明显的争用。主键的上界会成为“热点”。因为所有的插入都发生在这里，所以并发插入可能导致间隙锁竞争。另一个热点可能是auto_increment锁机制；如果遇到这个问题，则可能需要考虑重新设计表或者应用，或者更改innodb_autonc_lock_mode配置。