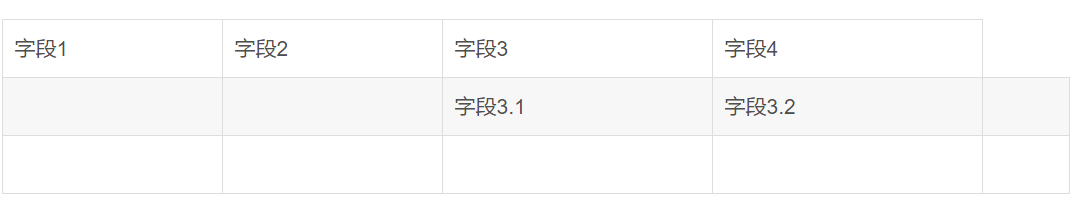
# 数据库原理

## 范式

**第一范式（1NF）无重复的列**

所谓第一范式（1NF）是指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。

反例：



**第二范式（2NF）属性完全依赖于主键 [ 消除部分子函数依赖 ]**

如果关系模式R为第一范式，并且R中每一个非主属性完全函数依赖于R的某个候选键， 则称为第二范式模式。

反例：

(学号, 课程名称) → (姓名, 年龄, 成绩, 学分)

因为存在如下关系：

(课程名称) → (学分)

(学号) → (姓名, 年龄)

**第三范式（3NF）属性不依赖于其它非主属性 [ 消除非主属性传递依赖 ]**

如果关系模式R是第二范式，且每个非主属性都不传递依赖于R的候选键，则称R为第三范式模式。

反例：

(学号) → (姓名, 年龄, 所在学院, 学院地点, 学院电话)

因为存在如下关系：

(学号) → (所在学院) → (学院地点, 学院电话)

**BCNF [ 消除主属性传递依赖 ]**

若关系模式R是第一范式，且每个属性都不传递依赖于R的候选键。

假设仓库管理关系表为StorehouseManage(仓库ID, 存储物品ID, 管理员ID, 数量)，且有一个管理员只在一个仓库工作；一个仓库可以存储多种物品。存在如下决定关系：

(仓库ID, 存储物品ID) →(管理员ID, 数量) (管理员ID, 存储物品ID) → (仓库ID, 数量)

还存在关键字段决定关键字段的情况。

(仓库ID) → (管理员ID) (管理员ID) → (仓库ID)。

**第四范式**

**表中不能包含一个实体的两个或多个互相独立的多值因子。**

反例：

职工表(职工编号，职工孩子姓名，职工选修课程)，在这个表中，同一个职工可能会有多个职工孩子姓名，同样，同一个职工也可能会有多个职工选修课程，即这里存在着多值事实，不符合第四范式。

## [事务及事务隔离级别](https://www.cnblogs.com/xrq730/p/5087378.html)

事务是访问数据库的一个操作序列，数据库应用系统通过事务集来完成对数据库的存取。**事务的正确执行使得数据库从一种状态转换为另一种状态**。

**原子性（Atomicity）**

一个事务内所有操作共同组成一个原子包，要么全部成功，要么全部失败。这是最基本的特性，保证了因为一些其他因素导致数据库异常，或者宕机。

**一致性（Consistency）**

原子性只保证了一个事务内的所有操作同一性，大家同生死，不会出现你死了，我还活着。但是，原子性并没有保证大家同一时刻一起生，一起死。计算机指令是有先后顺序的，这样就决定了一个事务的提交，会经历一个时间过程，那么如果事务提交进行到了一半，我读取了数据

库，会不会读到中间结果？为了防止这样的情况，数据库事务的一致性就规了事物提交前后，永远只可能存在事务提交前的状态和事务提交后的状态，从一个一致性的状态到另一个一致性状态，而不可能出现中间的过程态。也就是说事务的执行结果是量子化状态，而不是线性状态。

**隔离性（Isolation）**

事务的隔离性，基于原子性和一致性，因为事务是原子化，量子化的，所以，事务可以有多个原子包的形式并发执行，但是，每个事务互不干扰。

**持久性（Durability）**

持久性，当一个事物提交之后，数据库状态永远的发生了改变，这个事物只要提交了，哪怕提交后宕机，他也确确实实的提交了，不会出现因为刚刚宕机了而让提交不生效。

### 隔离级别

**Read uncommitted读未提交**

一个事务可以读取另一个未提交事务的数据。脏读。

**Read committed读提交**

就是一个事务要等另一个事务提交后才能读取数据。

例子：

程序员拿着信用卡去享受生活（卡里当然是只有3.6万），当他埋单时（程序员事务开启），收费系统事先检测到他的卡里有3.6万，就在这个时候！！程序员的妻子要把钱全部转出充当家用，并提交。当收费系统准备扣款时，再检测卡里的金额，发现已经没钱了（第二次检测金额当然要等待妻子转出金额事务提交完）。程序员就会很郁闷，明明卡里是有钱的…

若有事务对数据进行更新（UPDATE）操作时，读操作事务要等待这个更新操作事务提交后才能读取数据，可以解决脏读问题。但会出现一个事务范围内两个相同的查询却返回了不同数据，这就是不可重复读。

**Repeatable read重复读**

在开始读取数据（事务开启）时，不再允许修改操作。

例子：

程序员拿着信用卡去享受生活（卡里当然是只有3.6万），当他埋单时（事务开启，不允许其他事务的UPDATE修改操作），收费系统事先检测到他的卡里有3.6万。这个时候他的妻子不能转出金额了。接下来收费系统就可以扣款了。

不可重复读对应的是修改，即UPDATE操作。但是可能还会有幻读问题。

**幻读**

程序员某一天去消费，花了2千元，然后他的妻子去查看他今天的消费记录（全表扫描FTS，妻子事务开启），看到确实是花了2千元，就在这个时候，程序员花了1万买了一部电脑，即新增INSERT了一条消费记录，并提交。当妻子打印程序员的消费记录清单时（妻子事务提交），发现花了1.2万元，似乎出现了幻觉，这就是幻读。

幻读问题对应的是插入INSERT操作，而不是UPDATE操作。

**Serializable 序列化**

Serializable 是最高的事务隔离级别，在该级别下，事务串行化顺序执行，可以避免脏读、不可重复读与幻读。但是这种事务隔离级别效率低下，比较耗数据库性能，一般不使用。

大多数数据库默认的事务隔离级别是Read committed，比如Sql Server , Oracle。Mysql的默认隔离级别是Repeatable read。

### 事务隔离级别和锁的关系

我们都知道事务的几种性质，数据库为了维护这些性质，尤其是一致性和隔离性，一般使用加锁这种方式。同时数据库又是个高并发的应用，同一时间会有大量的并发访问，如果加锁过度，会极大的降低并发处理能力。所以对于加锁的处理，可以说就是数据库对于事务处理的精髓所在。

因为有大量的并发访问，为了预防死锁，一般应用中推荐使用一次封锁法，就是在方法的开始阶段，已经预先知道会用到哪些数据，然后全部锁住，在方法运行之后，再全部解锁。这种方式可以有效的避免循环死锁，但在数据库中却不适用，因为在事务开始阶段，数据库并不知道会用到哪些数据。

数据库遵循的是两段锁协议，将事务分成两个阶段，加锁阶段和解锁阶段。

* 加锁阶段：在该阶段可以进行加锁操作。在对任何数据进行读操作之前要申请并获得S锁（共享锁，其它事务可以继续加共享锁，但不能加排它锁），在进行写操作之前要申请并获得X锁（排它锁，其它事务不能再获得任何锁）。加锁不成功，则事务进入等待状态，直到加锁成功才继续执行。
* 解锁阶段：当事务释放了一个封锁以后，事务进入解锁阶段，在该阶段只能进行解锁操作不能再进行加锁操作。



**下面详细解说：**

Read Uncommitted这种级别，数据库一般都不会用，而且任何操作都不会加锁，这里就不讨论了。

在Read Committed级别中，数据的读取都是不加锁的，但是数据的写入、修改和删除是需要加锁的。只有用到索引的列才会加行锁，否则会给整张表加锁。但在实际使用过程当中，MySQL做了一些改进，在MySQL Server过滤条件，发现不满足后，会调用unlock\_row方法，把不满足条件的记录释放锁 (违背了二段锁协议的约束)。这样做，保证了最后只会持有满足条件记录上的锁，但是每条记录的加锁操作还是不能省略的。

对于rr和rc

如果使用锁机制来实现这两种隔离级别，在可重复读中，该sql第一次读取到数据后，就将这些数据加锁，其它事务无法修改这些数据，就可以实现可重复读了。但这种方法却无法锁住insert的数据，所以当事务A先前读取了数据，或者修改了全部数据，事务B还是可以insert数据提交，这时事务A就会发现莫名其妙多了一条之前没有的数据，这就是幻读，不能通过行锁来避免。

介绍RR于序列化之前，先介绍一下悲观锁与乐观锁。

**悲观锁**

正如其名，它指的是对数据被外界修改持保守态度，因此，在整个数据处理过程中，将数据处于锁定状态。悲观锁的实现，往往依靠数据库提供的锁机制。在悲观锁的情况下，为了保证事务的隔离性，就需要一致性锁定读。读取数据时给加锁，其它事务无法修改这些数据。修改删除数据时也要加锁，其它事务无法读取这些数据。

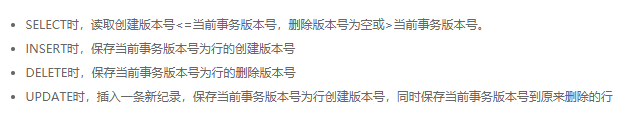
**乐观锁**

相对悲观锁而言，乐观锁机制采取了更加宽松的加锁机制。悲观锁大多数情况下依靠数据库的锁机制实现，以保证操作最大程度的独占性。但随之而来的就是数据库性能的大量开销，特别是对长事务而言，这样的开销往往无法承受。

乐观锁，大多是基于数据版本（ Version ）记录机制实现。即为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是通过为数据库表增加一个 “version” 字段来实现。读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据。

在InnoDb 的 RR级别下：

会在每行数据后添加两个额外的隐藏的值来实现MVCC，这两个值一个记录这行数据何时被创建，另外一个记录这行数据何时过期（或者被删除）。 在实际操作中，存储的并不是时间，而是事务的版本号，每开启一个新事务，事务的版本号就会递增。



事务的隔离级别其实都是对于读数据的定义，但到了这里，就被拆成了读和写两个模块来讲解。这主要是因为MySQL中的读，和事务隔离级别中的读，是不一样的。

InnoDb使用MVCC之后是可以解决幻读的。虽然让数据变得可重复读，但我们读到的数据可能是历史数据，是不及时的数据，不是数据库当前的数据。这在一些对于数据的时效特别敏感的业务中，就很可能出问题。对于这种读取历史数据的方式，我们叫它快照读 (snapshot read)，而读取数据库当前版本数据的方式，叫当前读 (current read)。

在MVCC中：

快照读：就是select

select \* from table ....;

当前读：特殊的读操作，插入/更新/删除操作，属于当前读，处理的都是当前的数据，需要加锁。

select \* from table where ? lock in share mode;

select \* from table where ? for update;

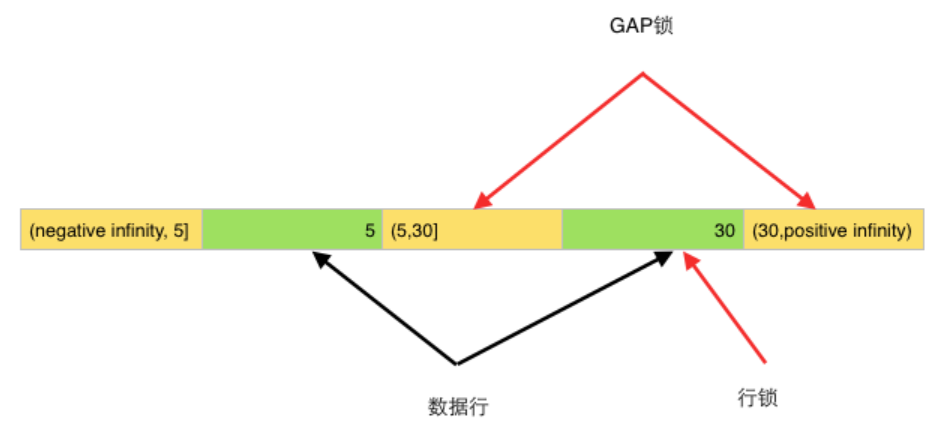
insert;

update ;

delete;

事务的隔离级别实际上都是定义了当前读的级别，MySQL为了减少锁处理（包括等待其它锁）的时间，提升并发能力，引入了快照读的概念，使得select不用加锁。而update、insert这些“当前读”，就需要另外的模块来解决了。

**间隙锁**



不仅用行锁，锁住了相应的数据行。同时也在两边的区间，（5,30]和（30，positive infinity），都加入了gap锁。这样事务B就无法在这个两个区间insert进新数据。但是可能会影响正常插入。如果使用的是没有索引的字段，那么会对全表加gap锁。正常的RR级别就行锁即可以实现。

**Serializable**

这个级别很简单，读加共享锁，写加排他锁，读写互斥。使用的悲观锁的理论，实现简单，数据更加安全，但是并发能力非常差。如果你的业务并发的特别少或者没有并发，同时又要求数据及时可靠的话，可以使用这种模式。

## 索引

### 定义及类型

数据库[索引](https://link.jianshu.com/?t=http://baike.baidu.com/view/262241.htm)好比是一本书前面的目录，能加快数据库的查询速度。[索引](https://link.jianshu.com/?t=http://baike.baidu.com/view/262241.htm)是对数据库表中一个或多个列（例如，employee表的姓氏(lname)列）的值进行排序的结构。

**优缺点：**

优点：

1.大大加快数据的检索速度;

2.创建唯一性索引，保证数据库表中每一行数据的唯一性;

3.加速表和表之间的连接;

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

缺点：

1.索引需要占用数据表以外的物理存储空间

2.创建索引和维护索引要花费一定的时间

3.当对表进行更新操作时，索引需要被重建，这样降低了数据的维护速度。

从功能上看，索引可分为**普通索引**、**唯一索引、主键索引：**

**普通索引：**普通索引是基本索引类型，允许在定义索引的列中插入重复值和空值

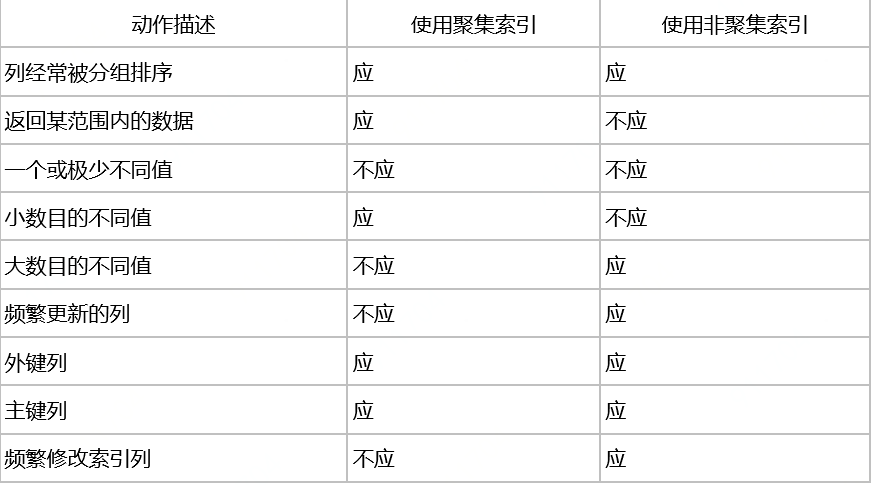
**唯一索引：**表明此索引的每一个索引值只对应唯一的数据记录，允许空值，对于单列惟一性索引，这保证单列不包含重复的值。对于多列惟一性索引，保证多个值的组合不重复。

**主键索引：**主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一且不为空。

从存储结构上看，分为**聚集索引**和**非聚集索引**：

**聚集索引：**该索引中键值的逻辑顺序决定了表中相应行的物理顺序。（因此一个表只能包含一个聚集索引）。

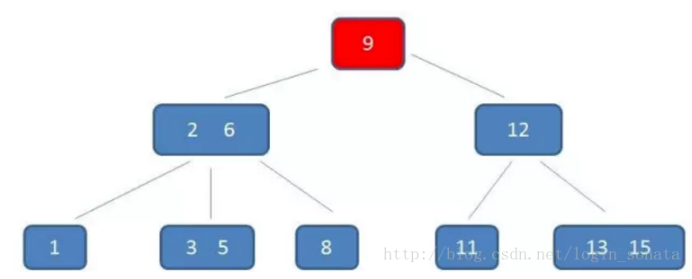
**非聚集索引：**该索引中索引的逻辑顺序与磁盘上行的物理存储顺序不同。



### B树和B+树

**一个M阶的b树具有如下几个特征**：

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]，向上取整；
4. 非叶子结点的关键字个数=儿子数-1；
5. 所有叶子结点位于同一层；
6. k个关键字把节点拆成k+1段，分别指向k+1个儿子，同时满足查找树的大小关系

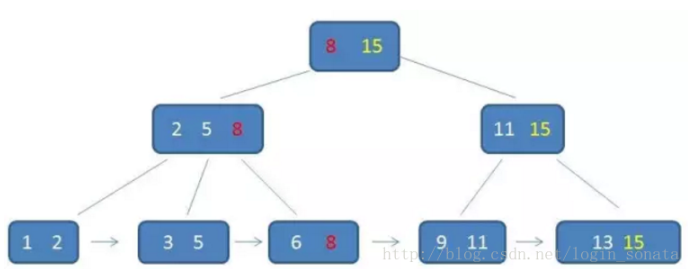


**有关b树的一些特性，注意与后面的b+树区分：**

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；

**b+树，是b树的一种变体，查询性能更好。m阶的b+树的特征**：

1. 有n棵子树的非叶子结点中含有n个关键字（b树是n-1个），这些关键字不保存数据，只用来索引，所有数据都保存在叶子节点（b树是每个关键字都保存数据）。
2. 所有的叶子结点中包含了全部关键字的信息，及指向含这些关键字记录的指针，且叶子结点本身依关键字的大小自小而大顺序链接。
3. 所有的非叶子结点可以看成是索引部分，结点中仅含其子树中的最大（或最小）关键字。
4. 通常在b+树上有两个头指针，一个指向根结点，一个指向关键字最小的叶子结点。
5. 同一个数字会在不同节点中重复出现，根节点的最大元素就是b+树的最大元素。



b+树相比于b树的**查询优势**：

1. b+树的中间节点不保存数据，所以磁盘页能容纳更多节点元素，更“矮胖”；
2. b+树查询必须查找到叶子节点，b树只要匹配到即可不用管元素位置，因此b+树查找更稳定（并不慢）；
3. 对于范围查找来说，b+树只需遍历叶子节点链表即可，b树却需要重复地中序遍历。

### 建立原则及失效情况

**建立的几大原则：**

1. 最左前缀匹配原则

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。

1. 尽量选择区分度高的列作为索引
2. 索引列不能参与计算
3. 在经常需要搜索的列上，经常用在连接的列上，根据范围进行搜索的列上，经常需要排序的列上，经常使用在WHERE子句中的列上应当建立索引。

**失效情况：**

1.WHERE字句的查询条件里有不等于号或or或null，not null（如：WHERE column!=...）

2.WHERE字句的查询条件里使用了函数的列（如：WHERE DAY(column)=...）

3.WHERE子句的查询条件里使用了比较操作符LIKE前置通配符%（如：LIKE "%ABC"）

4.在ORDER BY操作中，排序的列同时也在WHERE中时，MYSQL将无法使用索引

5.对于多列索引，不是使用的第一部分(第一个)，则不会使用索引（最左匹配）

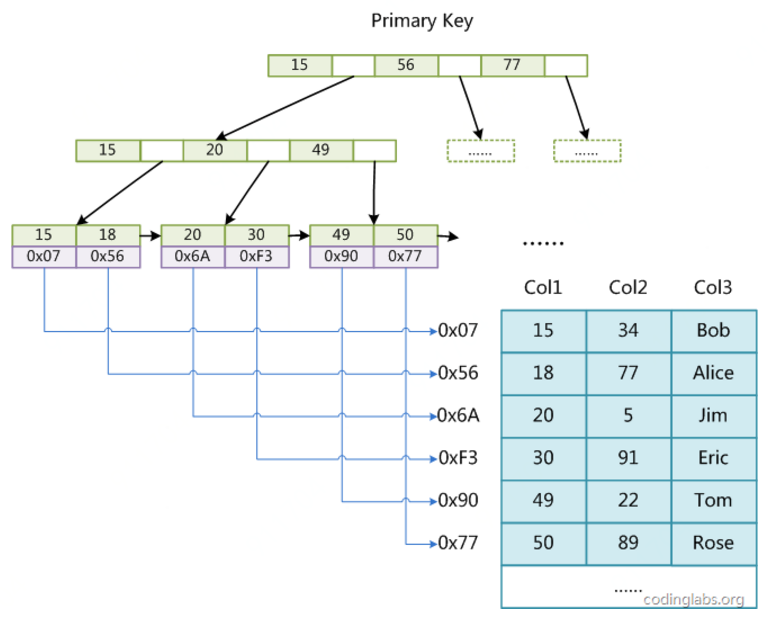
# Mysql

## InnoDb和MyIsam引擎对比

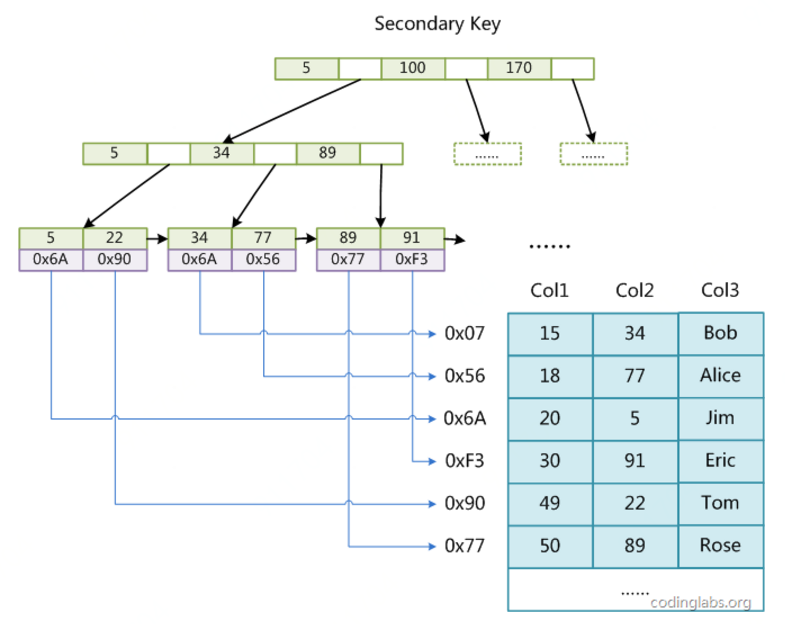
1. 事务。MyISAM类型不支持事务处理等高级处理，而InnoDB类型支持，提供事务支持已经外部键等高级数据库功能。InnoDB的默认隔离级别是可重复读。
2. 性能主题。MyISAM类型的表强调的是性能，其执行数度比InnoDB类型更快。
3. 行数保存。InnoDB 中不保存表的具体行数，也就是说，执行select count() fromtable时，InnoDB要扫描一遍整个表来计算有多少行，但是MyISAM只要简单的读出保存好的行数即可。
4. 锁的支持。MyISAM只支持表锁。InnoDB支持表锁、行锁 行锁大幅度提高了多用户并发操作的新能。但是InnoDB的行锁，只是在WHERE的主键是有效的，非主键的WHERE都会锁全表的。
5. 其他。MyISAM支持全文索引（FULLTEXT）、压缩索引，InnoDB不支持；MyISAM的索引和数据是分开的，并且索引是有压缩的，内存使用率就对应提高了不少。能加载更多索引，而Innodb是索引和数据是紧密捆绑的。
6. 索引实现

**MyISAM索引实现**

使用B+Tree作为索引结构，叶节点的data域存放的是数据记录的地址。（非聚集）

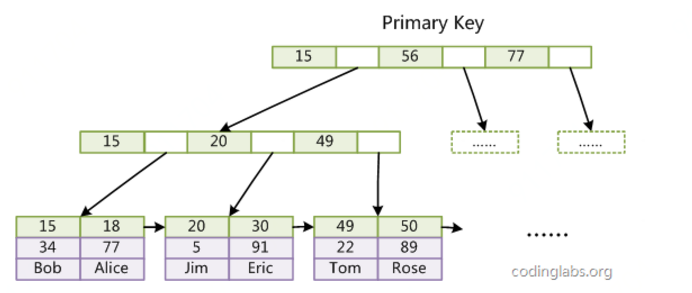


这里设表一共有三列，假设我们以Col1为主键，则图8是一个MyISAM表的主索引（Primary key）示意。可以看出MyISAM的索引文件仅仅保存数据记录的地址。在MyISAM中，主索引和辅助索引（Secondary key）在结构上没有任何区别，只是主索引要求key是唯一的，而辅助索引的key可以重复。如果我们在Col2上建立一个辅助索引，则此索引的结构如下图所示：



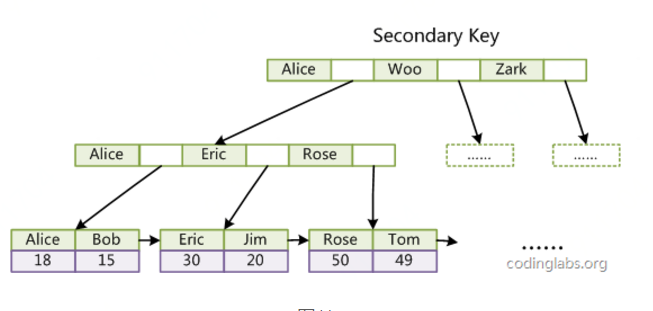
**InnoDB索引实现**

InnoDB也使用B+Tree作为索引结构，但InnoDB的数据文件本身就是索引文件。MyISAM索引文件和数据文件是分离的，索引文件仅保存数据记录的地址。而在InnoDB中，表数据文件本身就是按B+Tree组织的一个索引结构，这棵树的叶节点data域保存了完整的数据记录。索引的key是数据表的主键，因此InnoDB表数据文件本身就是主索引。



因为InnoDB的数据文件本身要按主键聚集，所以InnoDB要求表必须有主键（MyISAM可以没有），如果没有显式指定，则MySQL系统会自动选择一个可以唯一标识数据记录的列作为主键，如果不存在这种列，则MySQL自动为InnoDB表生成一个隐含字段作为主键，这个字段长度为6个字节，类型为长整形。

图为定义在Col3上的一个辅助索引：



聚集索引这种实现方式使得按主键的搜索十分高效，但是辅助索引搜索需要检索两遍索引：首先检索辅助索引获得主键，然后用主键到主索引中检索获得记录。