## 一．Mysql体系结构与存储引擎

### 1.1数据库和数据库实例

数据库是物理操作系统文件或其他类型文件的集合，是文件的集合，

数据库实例是应用程序，由数据库后台进程、线程、共享区域组成，数据数据库实例才是真正用来操作数据库文件的。

库实例才是真正用来操作数据库文件的。

Mysql是一个单进程多线程的数据库，即mysql实例在系统上表现为一个进程。

一个数据库实例就是一个进程而数据库可以有多个客户端连接，每个客户端的连接就是一个线程，这就是所谓的

数据库：物理操作系统文件或其他形式文件两类型的集合。在mysql中，数据库文件，可以是frm、myd、myi、idb结尾的文件

数据库实例：由数据库后台进程、线程以及一个共享内存区域组成

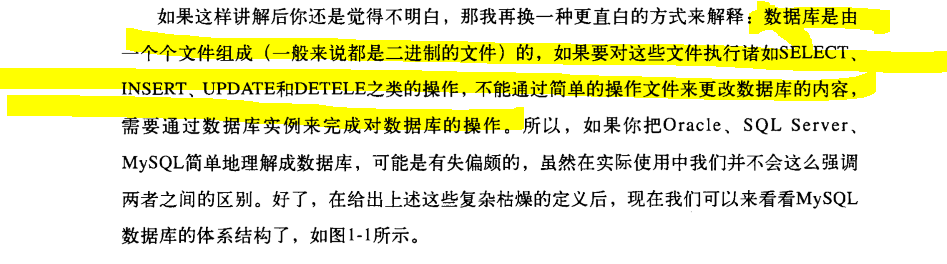
共享内存可以被运行的后台进程/线程分享。数据库实例才是真正用来操作数据库文件的。

在mysql中数据库实例和数据库是一一对应的，但在集群的情况下一个数据库可以被多个实例使用。

当启动实例时，mysql数据库会读取配置文件，根据配置文件的参数来启动数据库实例

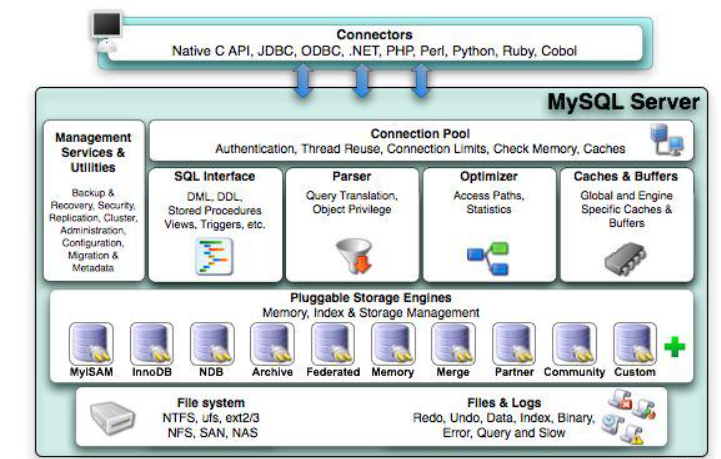
数据库是文件的集合，是依照某种数据模型组织起来并存放在二级存储器中的数据集合

数据库实例是应用程序，是位于用户与操作系统之间的一层数据管理软件，用户对数据库数据的任何操作都是在数据库实例下进行的。



### 1.2Mysql体系结构

#### mysql体系结构图



连接池组件

管理服务和工具组件

SQL接口组件

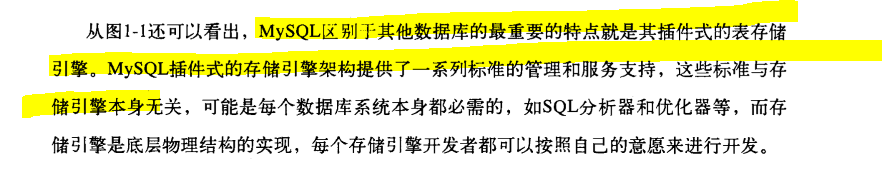
查询分析器组件

优化器组件

缓冲组件

插件式存储引擎

物理文件





##### 网络连接层

负责连接、认证等工作。

每个客户端的连接都对应着服务器上的一个线程。

由于数据库的连接是有上限的这和系统配置有关，因此mysql也有着自己管理客户端连接的机制，线程池。

###### 线程池

Mysql会有一个线程池，避免每一个客户端连接都销毁一个线程。

创建线程池，是因为每个客户端的连接都是一个单独的线程执行，当并发量打的时候，没有一个客户端连接就要新建一个线程，并且要负责摧毁这个线程。会销毁系统很大的资源。有个线程池就不用为每个用户创建销毁一个线程

###### Mysql线程池工作原理

##### 服务层

服务层是mysql的核心，负责sql的执行、分析、优化，查询缓存、跨存储引擎的实现

###### Sql的执行流程

##### 存储引擎层

负责mysql数据的提取与存储，mysql通过api与存储引擎通信

存储引擎针对于表不针对于库

##### 系统文件层

将数据库的数据存储在文件系统之上，并完成与存储引擎的交互。

##### 服务层与存储引擎层的交互

https://stor.51cto.com/art/202004/614479.htm

### 1.3Mysql存储引擎

#### 1.31 InnoDB存储引擎

### 1.5连接Mysql

连接mysql操作是连接连接进程和mysql数据库实例进行通信

## 5.索引与算法

### 5.1innodb存储索引

Innodb常见索引：B+树索引、哈希索引

Innodb存储引擎支持的哈希索引是自适应的，innodb存储引擎会根据表的使用情况自动为表生成哈希索引。

B+树索引是传统意义上的索引，是目前关系型数据库中最常用的、最有效的索引，其构造类似于二叉树，根据键值快速找到数据

B+树是由平衡二叉树演化而来，B+树不是二叉树

### 5.2二分法查找

略

### 5.3平衡二叉树

### 5.4B+树

在理解B+树之前先要理解B-树

数据库为什么要用树形结构存储，因为树的查询效率高而且可以保持有序，

那为什么没有用二叉排序树来实现呢，二叉查找树查询的复杂度是O（logN）,查找速度与比较次数是最小的，但是数据库的索引是存储在磁盘上的，当数据量比较大的时候，索引可能有几个G甚至更多，不可能吧整个索引全部加载到内存中，只能逐一加载磁盘页，每个磁盘页对应着索引树的节点，



从图中可以看出磁盘IO次数和树的高度有关，因此为了减少磁盘IO次数，我们需要把原来高瘦的树结构变成矮胖的

那么什么是B树呢

#### B树 B-tree

很多人读成B- b减树 是错的

##### 定义

一个m阶的B树具有如下几个特征：

1.根结点至少有两个子女。

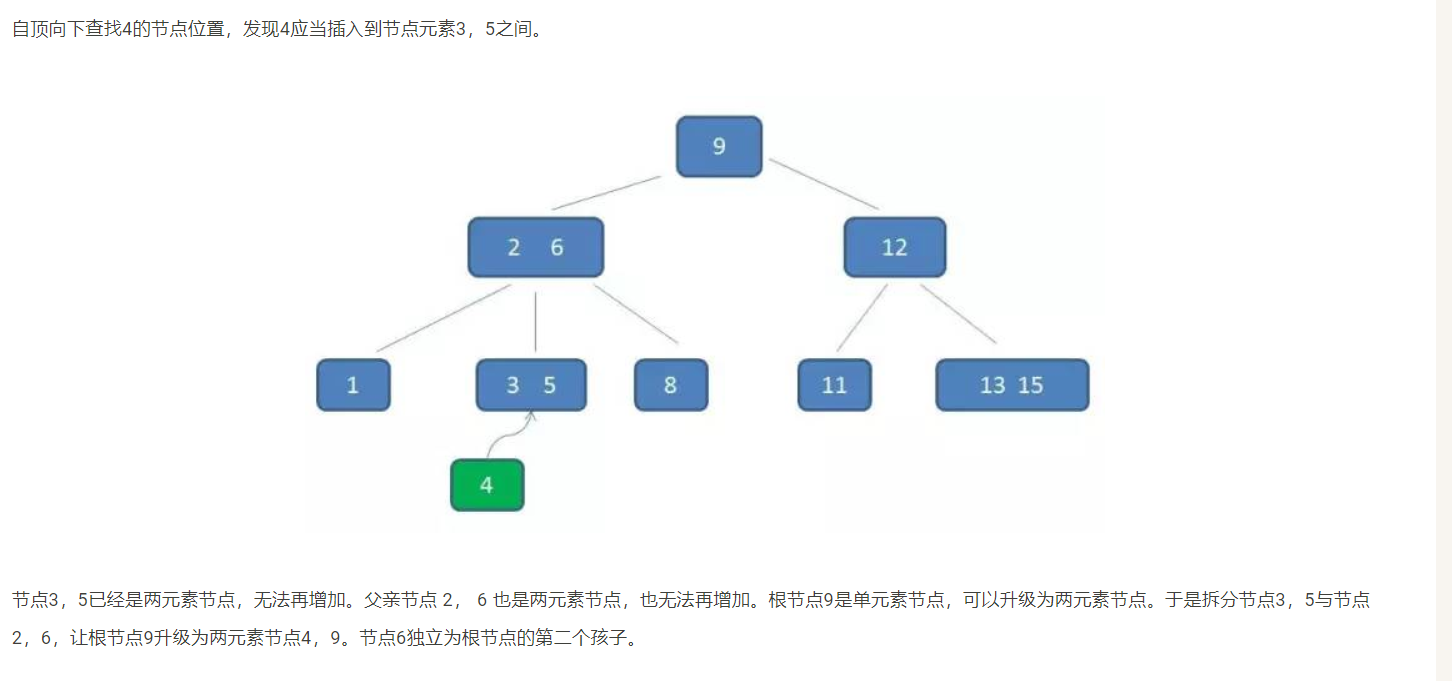
2.每个中间节点都包含k-1个元素和k个孩子，其中 m/2 <= k <= m

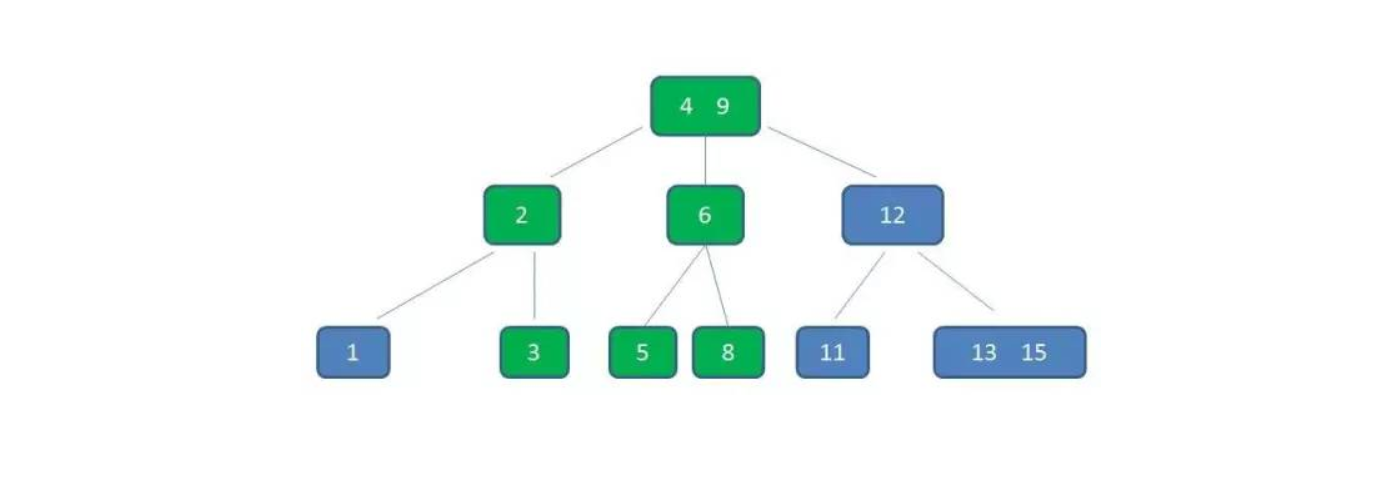
3.每一个叶子节点都包含k-1个元素，其中 m/2 <= k <= m

4.所有的叶子结点都位于同一层。

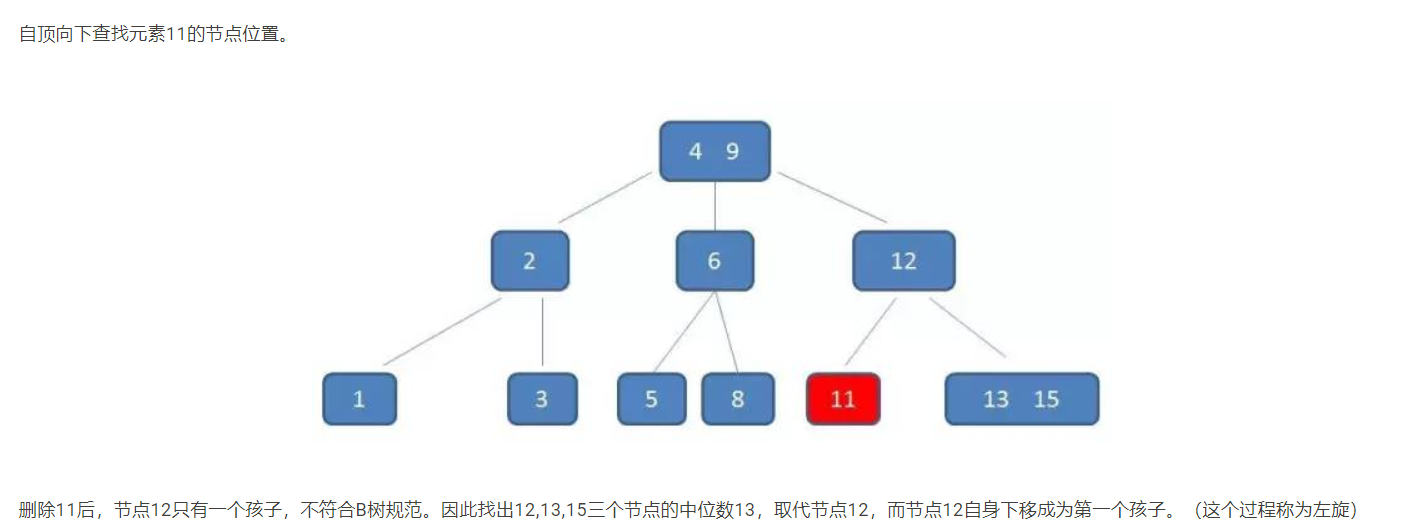
5.每个节点中的元素从小到大排列，节点当中k-1个元素正好是k个孩子包含的元素的值域分划。

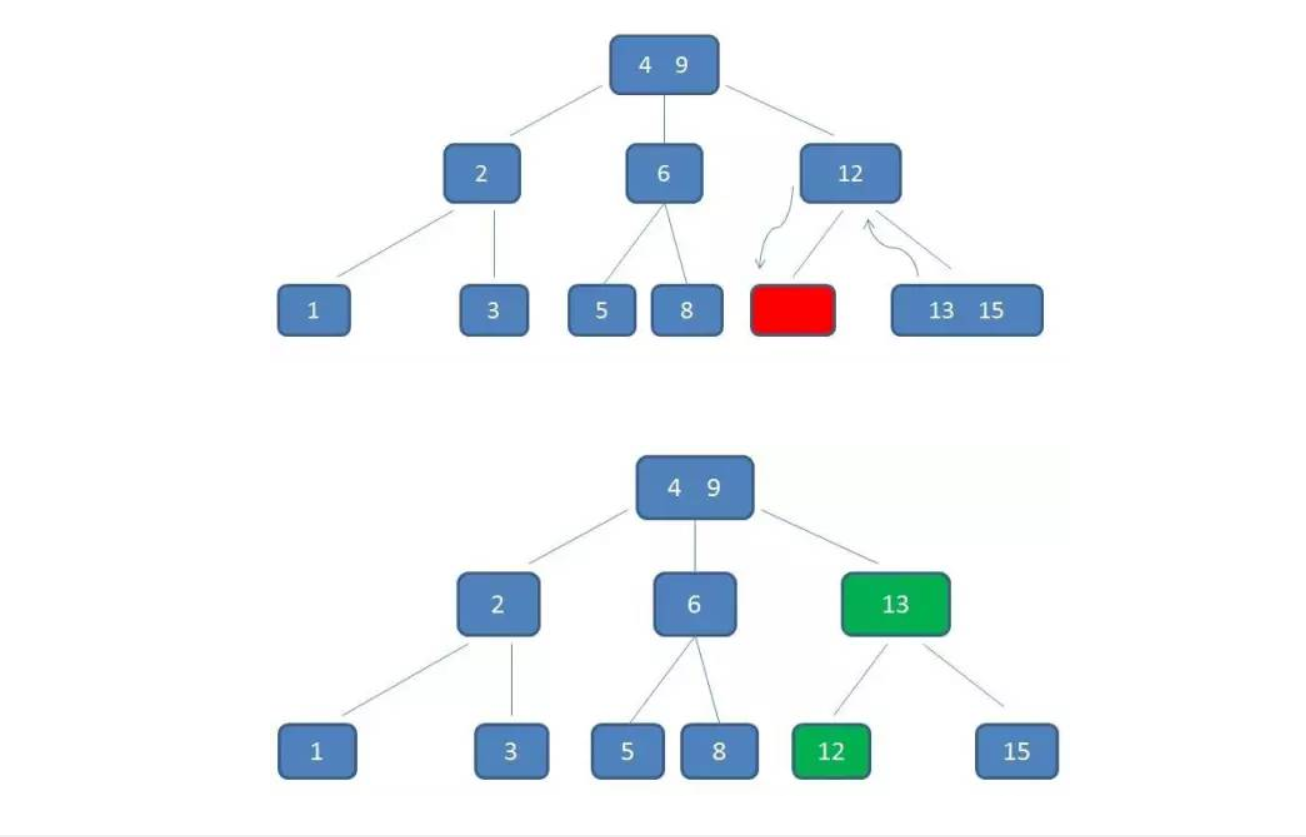
##### 插入新节点





##### 删除节点





##### 应用

B树主要用于文件系统以及部分数据库索引，如关系型数据库MongoDB

##### 代码实现

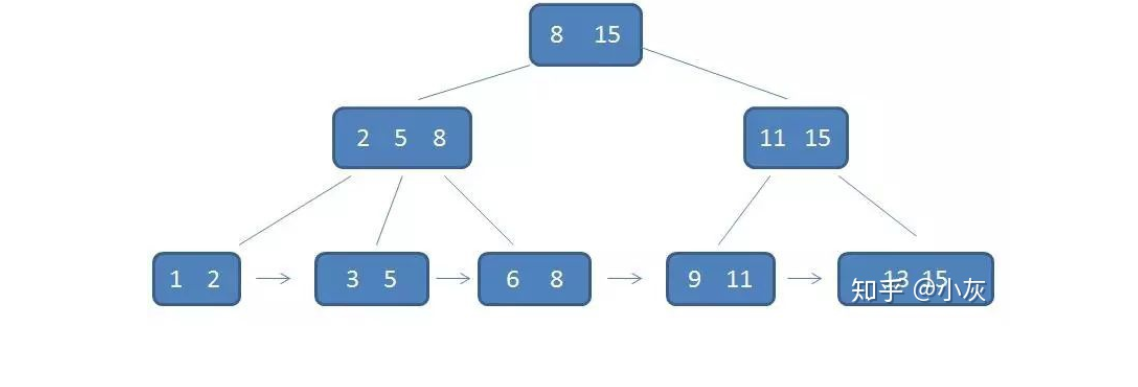
#### B+树

##### 定义

1.有k个子树的中间节点包含有k个元素（B树中是k-1个元素），每个元素不保存数据，只用来索引，所有数据都保存在叶子节点。

2.所有的叶子结点中包含了全部元素的信息，及指向含这些元素记录的指针，且叶子结点本身依关键字的大小自小而大顺序链接。

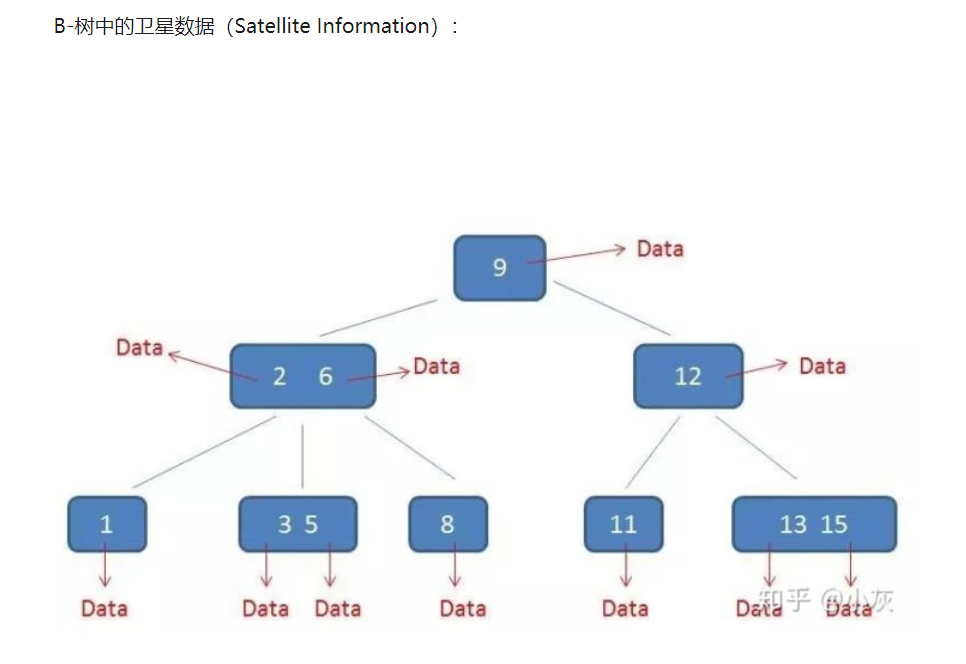
3.所有的中间节点元素都同时存在于子节点，在子节点元素中是最大（或最小）元素。



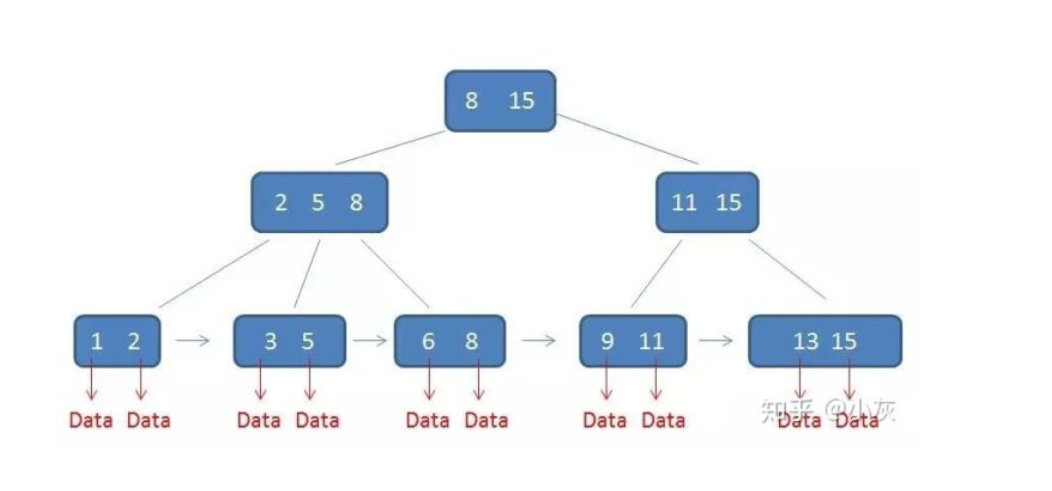
##### 卫星数据

卫星数据是指索引元素所指向的数据记录

在B树中无论是中间节点还是叶子节点都带有卫星数据



在B+树中只用叶子结点带有卫星数据，其余中间节点仅仅是索引，没有任何关联数据



在聚集索引中，叶子节点直接包含卫星数据，在非聚集索引中，叶子节点带有指向卫星数据的指针

因为B+树的中间节点没有卫星数据，所以同样大小的磁盘可以容纳更多的节点元素，这也就意味着在数据量相同的情况下，B+树的结构比B树更加矮胖磁盘的IO次数更少。

B+树比B树的优势：

磁盘IO次数少，查询性能稳定，范围查询简便

#### 二叉查找树（二叉排序树）

##### 定义

若它的左子树不空，则左子树上的所有节点的值均小于根节点的值

若它的右子树不空，则柚子树上的所有节点的值均大于根节点的值

它的左右子树都是二叉排序树

##### 代码实现

###### 二差排序树的插入

### 聚簇索引(聚集索引)

InnoDB存储引擎表是索引组织表，表中的数据是按照主键顺序存放，而聚集索引就是按照每张表的主键构造B+树，并且叶子节点中存放着整张表的行记录数据，因此聚集索引的叶子节点也叫数据页。

聚集索引的这个特性决定了索引组织表中的数据也是索引的一部分，和b+树一样，每个数据页都通过一个双向链表来连接。

由于实际的数据页只能按照一颗B+树进行排序，因此每张表只能拥有一个聚集索引

### 非聚集索引

非聚集索引叶子节点不包括行的全部数据，叶子节点除了包含键值以外，每个页级别的索引

行中还包含一个书签(即主键值或者指向数据的指针 )，用于告诉innoDB引擎，哪里可以找到与索引相对应的行数据

因为innoDB存储引擎表是索引组织表，因此innoDb存储引擎的非聚集索引的书签就是相应行数据的聚集索引键。

非聚集索引的存在并不影响数据在聚集索引中，因此每张表上可以有多个非聚集索引

当通过非聚集索引来寻找数据时，innoDB存储引擎会遍历非聚集索引并通过叶子节点存放的书签找到指向主键索引的主键，然后再通过主键来找到一个完整的行记录

### 索引的创建

Create index 索引名 on 表名（字段名(长度)）

Create index indexName on table(username(10))

索引使用列名称的前10个字符。

因为多数名称的前10个字符通常不同，所以此索引不会比使用列的全名创建的索引速度慢很多

目前mysql数据库存在的一个普遍问题是，所有对于索引的添加或者删除操作，mysql数据库是先创建一张新的临时表，然后把数据导入临时表，删除原表，再把临时表重名为原来的表名。

因此对于一个很大的表，添加和删除索引需要很长的时间。

### 联合索引

在表的多个字段加的索引

Create index 索引名 on 表名（字段名(长度),字段名2（长度））

Create index indexName on table(username(10),password(2))

#### 最左匹配

当创建了Create index indexName on table (col1,col2,col3)这样一个联合索引，实际上相当于创建了（col1）(col1,col2),(col1,col2,col3)三个索引

Select \* from test where col=1 and col2=2 and col4=4;当执行这样一个查询时根据最左匹配原则会使用索引（col1,col2）进行匹配。

#### 联合索引好处

##### 减少开销。

建一个联合索引(col1,col2,col3)，实际相当于建了(col1),(col1,col2),(col1,col2,col3)三个索引。每多一个索引，都会增加写操作的开销和磁盘空间的开销。对于大量数据的表，使用联合索引会大大的减少开销！

##### 覆盖索引。

对联合索引(col1,col2,col3)，如果有如下的sql: select col1,col2,col3 from test where col1=1 and col2=2。**那么MySQL可以直接通过遍历索引取得数据，而无需回表，这减少了很多的随机io操作。**减少io操作，特别的随机io其实是dba主要的优化策略。所以，在真正的实际应用中，覆盖索引是主要的提升性能的优化手段之一。

##### 效率高。

**索引列越多，通过索引筛选出的数据越少。**有1000W条数据的表，有如下sql:select from table where col1=1 and col2=2 and col3=3,假设假设每个条件可以筛选出10%的数据，如果只有单值索引，那么通过该索引能筛选出1000W10%=100w条数据，然后再回表从100w条数据中找到符合col2=2 and col3= 3的数据，然后再排序，再分页；如果是联合索引，通过索引筛选出1000w10% 10% \*10%=1w，效率提升可想而知！

#### Mysql查询优化器

使用explain关键字，查看sql的执行情况

explain select \* from user;

#### mysql存在索引但不能使用索引的典型场景

1. 当mysql分析全表扫描比使用快的时候不适用索引
2. 用or分隔开的条件，如果or前的条件中的列有索引，or后的条件没有索引，不会用到索引
3. 使用联合索引时，索引列不是复合索引的第一部分

复合索引为(key1,key2),则查询select \* from table\_name where key2='b';将不会使用索引

1. like以’%’开始，该列上的索引不被使用

select \* from table\_name where key1 like '%a'

1. 列参与了数学运算或者函数
2. 使用不等于查询<>，not in 无法使用索引

此章节未看完

## 6.innoDB锁

Innodb自动加锁的部分

### 为什么要用锁

最大程度利用数据的并发访问，确保每个用户能以一致方式读取和修改数据

### 定义

确保每个用户可以能以一致性方式读取和修改数据

锁机制用于管理对共享资源的并发访问

### 创建语法

Select … lock in share mode //共享锁

Select … for update //排他锁

### InnoDB存储引擎中的锁

#### 锁类型

锁分为行级锁和表级锁

##### 共享锁(读锁) (行级锁)

//其他事务可以读不能写

SLock,允许事务读一行数据

##### 排他锁(写锁)(行级锁)

//其他事务不能读也不能写

XLock,允许事务删除或者更新一行数据

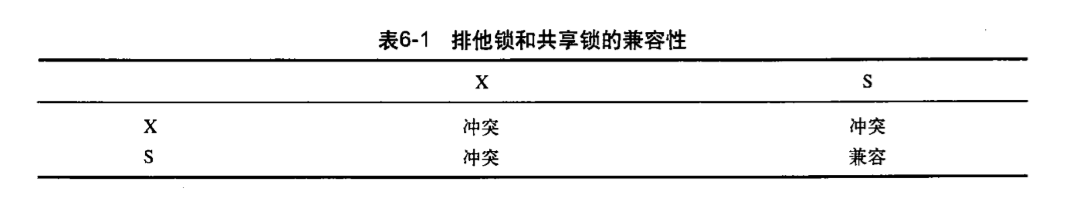
##### 锁兼容

当一个事务已经获得行r的共享锁，其他事务也可以立即获得行r的共享锁

##### 锁不兼容

当一个事务A已经获得行r的共享锁，其他事务要想获得行r上的排他锁就要等待事务A释放行r上的共享锁

##### 排他锁与共享锁的兼容性



##### 多粒度锁定

InnoDb存储引擎支持多粒度锁，这种锁允许在行级上的锁和表级上的锁同时存在。

##### 意向锁(表级锁)

为了在不同粒度上进行加锁操作，意向锁是表级别的锁，其设计目的是为了在一个事务中下一行将被请求的锁的类型。

##### 意向锁设计目的

在下一个事务中揭示下一行将被请求的锁的类型

##### 意向锁类型(表级锁)

###### 意向共享锁

IS Lock 事务想要获得一个表中某几行的共享锁

###### 意向排他锁

IX lock 事务想要获得一个表中某几行的排他锁

#### 一致性的非锁定行读

指InnoDB存储引擎通过行多版本控制的方式来读取当前执行时间数据库行中的数据，如果读取的行正在执行DELETE、UPDATE操作，这时读取操作不会因此等待行上锁的释放

假设数据库中存在id=1的数据 我们开启一个事务查询id=1的数据会返回一条数据

若果这时候再开一个会话，将id=1的数据的id修改为id=3，这时候在第一个事务中查询依然是会查出 id=1的一条数据，如果另一个事务提交了，在次查询第一个事务 则查询不到数据

#### 非锁定读机制

不需要等待访问的行上X锁的释放

#### 锁的创建

Select ..for update 加X锁 排他锁

Select ..lock in share mode 加S锁 共享锁

#### 自增长和锁

在innoDB存储引擎的内存结构中，对每个含有自增长测值得表都有一个自增长计数器，当含有自增长计数器的表进行插入操作时，这个计数器会被初始化。

插入操作会依据自增长的计数器值加1赋予自增长列。这个实现方式叫做AUTO-INC locking。

这种锁其实采用一种特殊的表锁机制，为了提高插入的性能，锁不是在事务完成之后才释放的，而是在完成对自增长值插入的SQL语句后立即释放

自增长值的列的并发插入性能差，必须等待前一个插入的完成，

### 锁算法

#### Record Lock

单个行上的锁

#### Gap Lock(间隙锁)

锁定一个范围，并且锁定记录本身

#### Next-Key Lock

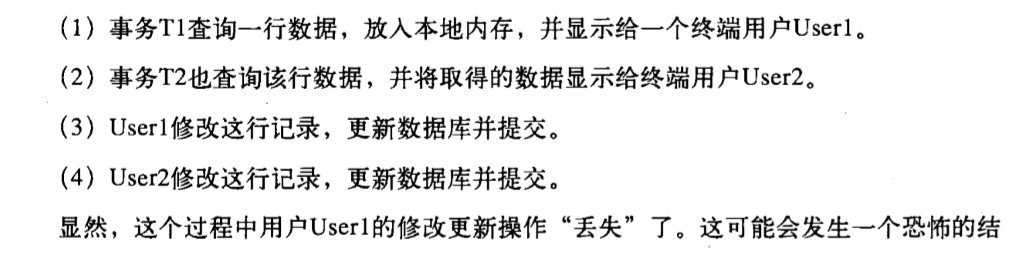
Gap Lock+Record Lock,锁定一个范围，并且锁定记录本身

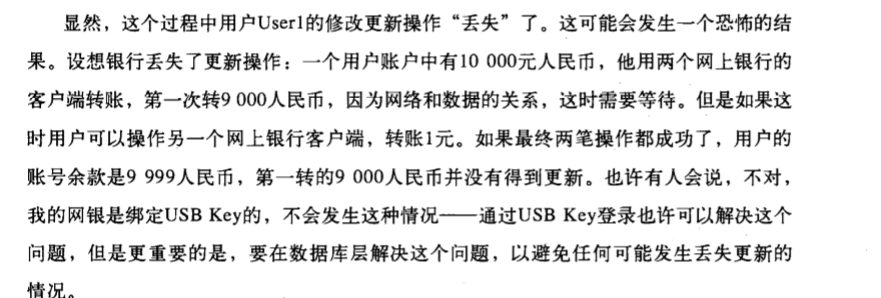
InnoDB 对于行的查询采用 next-key Lock

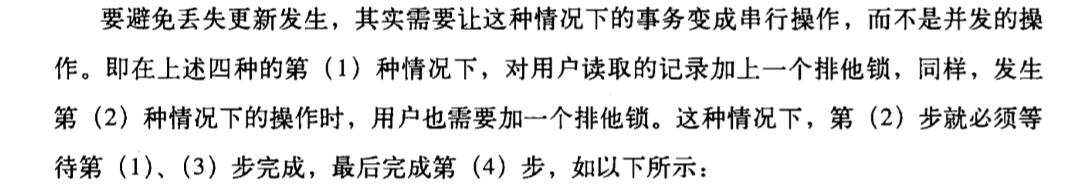
### 锁问题

通过锁可以实现事务的隔离性要求，锁提高了并发，带来了以下三个问题

#### 丢失更新







#### 脏读

**脏页：**

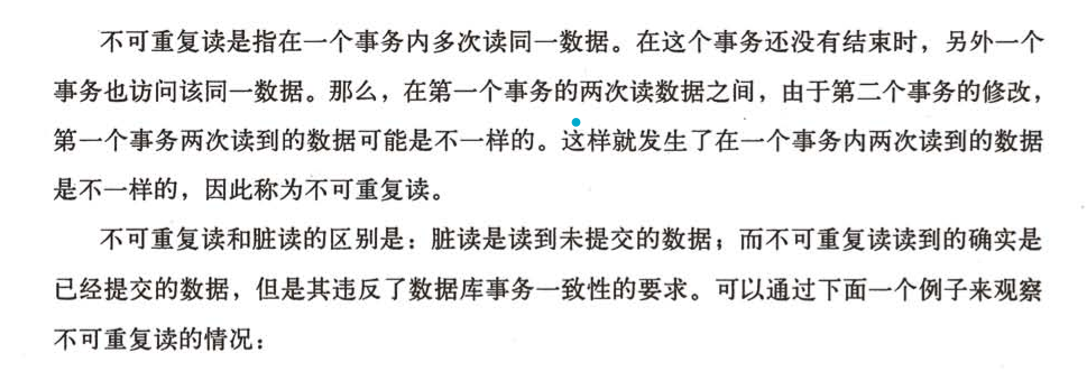
指的是在缓冲池中已经被修改的页，但还没刷新到磁盘

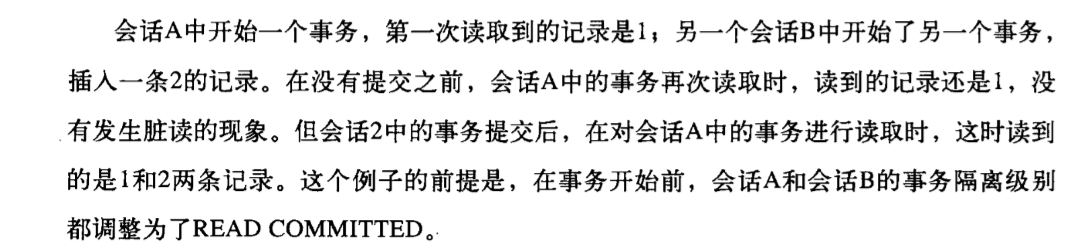
脏数据：

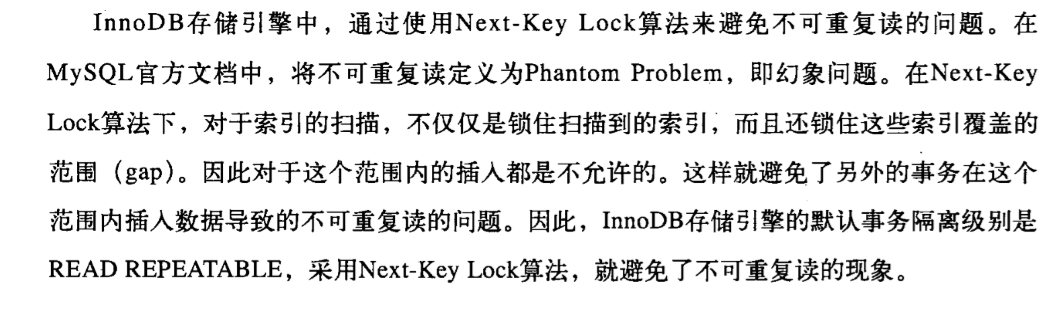
是指在缓冲池中被修改的数据，但是还没有被提交

**脏读指的是在不同的事务下，可以读到另外事务未提交的数据，即可以读到脏数据**

#### 不可重复读







### 阻塞

#### 什么是阻塞

由于不同锁之间的兼容性，某些时刻，一个事务的锁需要等待另一个事务中的锁释放它所占用的资源。这就是阻塞，阻塞不是坏事，是为了保证事务可以并发和正常的运行

### 死锁

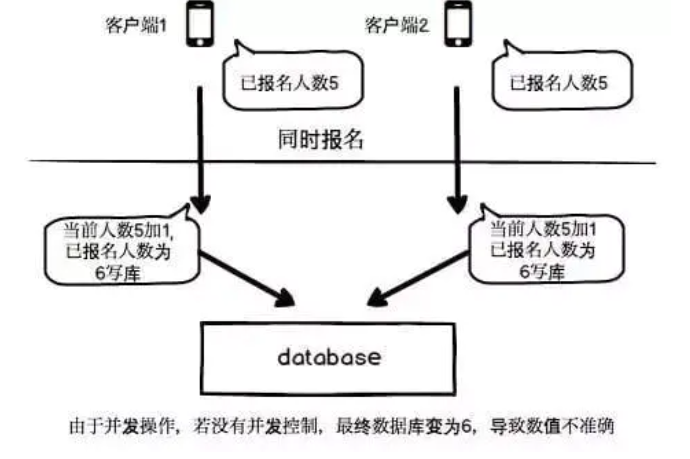
死锁只发生于并发的情况



### 并发控制

参考：<https://www.jianshu.com/p/d2ac26ca6525>

当程序中可能出现并发的情况时，就需要通过一定的手段来保证在并发情况下数据的准确性，通过这种手段保证了当前用户和其他用户一起操作时，所得到的结果和他单独操作时的结果是一样的。这种手段就叫做并发控制。并发控制的目的是保证一个用户的工作不会对另一个用户的工作产生不合理的影响



#### 并发控制的手段

##### 加锁

###### 悲观锁

当要对数据库中的一条数据进行修改的时候，为了避免同时被其他人修改，最好的办法就是直接对该数据进行加锁以防止并发。这种借助数据库锁机制，在修改数据之前先锁定，再修改的方式被称之为悲观并发控制。

悲观锁的排他性，排除的是本系统的其他事务和外部系统对事务的修改，因此一般借助数据库提供的锁机制实现

**坏处：**

但是在效率方面，处理加锁的机制会让数据库产生额外的开销，还有增加产生死锁的机会。另外还会降低并行性，一个事务如果锁定了某行数据，其他事务就必须等待该事务处理完才可以处理那行数据。

###### 乐观锁

乐观锁假设数据一般情况下不会造成冲突，所以在数据进行提交更新的时候，才会正式对数据的冲突与否进行检测，如果发现冲突了，则返回给用户错误的信息，让用户决定如何去做。

乐观锁机制采取了更加宽松的加锁机制。乐观锁是相对悲观锁而言，也是为了避免数据库幻读、业务处理时间过长等原因引起数据处理错误的一种机制，但乐观锁不会刻意使用数据库本身的锁机制，而是依据数据本身来保证数据的正确性。

**乐观锁的实现**

**设置版本**

可以表中添加一个字段version用来判断是否被修改

修改则++;

假设A,B同时操作同一条记录，获取到的version都是2在修改之前先判断version的值，A先修改version变成3 这是B在修改 判断version为3则不执行

###### 场景

1. 乐观锁并未真正加锁，效率高。一旦锁的粒度掌握不好，更新失败的概率就会比较高，容易发生业务失败。
2. 悲观锁依赖数据库锁，效率低。更新失败的概率比较低。

##### 多版本控制MVCC

参考：<https://www.yuque.com/yinjianwei/vyrvkf/ri4ks7>

是数据库常用的处理读写冲突的手段，目的在于提高数据库高并发场景下的吞吐性能

## Mysql/innodb加锁分析

参考：<http://www.doc88.com/p-6921862510402.html> 何登成 mysql加锁处理分析

### Mvcc

Mvcc是基于多版本的并发控制协议。

好处：读不加锁，读写不冲突。

在mvcc多版本控制中，读操作分为两种，快照读、当前读。

#### 快照读

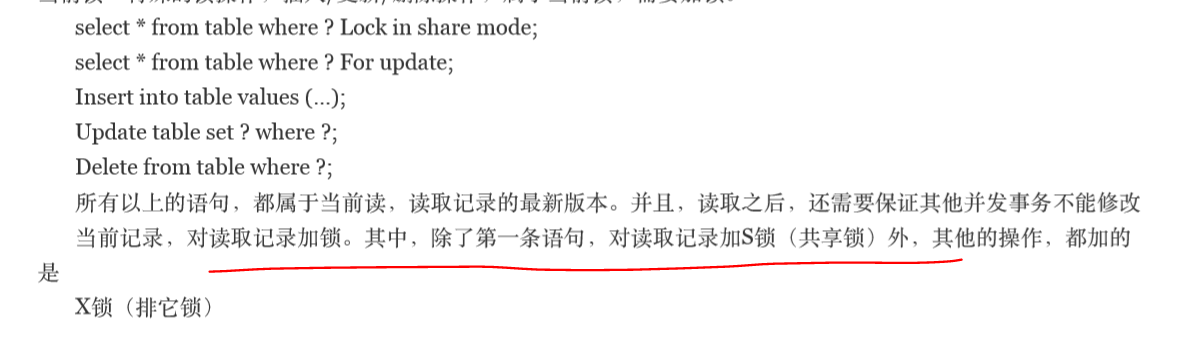
读取的记录是可见版本，不用加锁。

简单的Select属于快照读(也有例外)，不加锁

#### 当前读

读取的记录是最新版本，当前读返回的记录都会上锁，保证其他事务不会再并发修改这条记录

特殊的读操作，插入、更新、删除操作数据当前读，需要加锁。



图片的数据都是加了锁的，简单的select是快照读所以没有加锁这里手动加了共享锁 和排他锁，而insert update delete 操作属于当前读 所以innodb 引擎会自动加上排他锁，与开不开起事务无关。

#### 二阶段锁Two-phase-Locking

二阶段锁:所操作分为两个阶段，加锁阶段、解锁阶段，并保证加锁阶段和解锁阶段不相交。

二阶段锁在mysql中的实现



加锁阶段只加锁不解锁，解锁阶段只解锁。

#### 隔离级别

4中隔离级别：read uncommited, read commited,repeatable read,serializable

##### read uncommited

可以读取未提交记录，此隔离界别，不使用，忽略。

##### read commited(RC)

快照读忽略

针对当前读，RC隔离级别保证对读取到的记录加锁(记录锁),存在幻读现象。

##### repeatable read(RR)

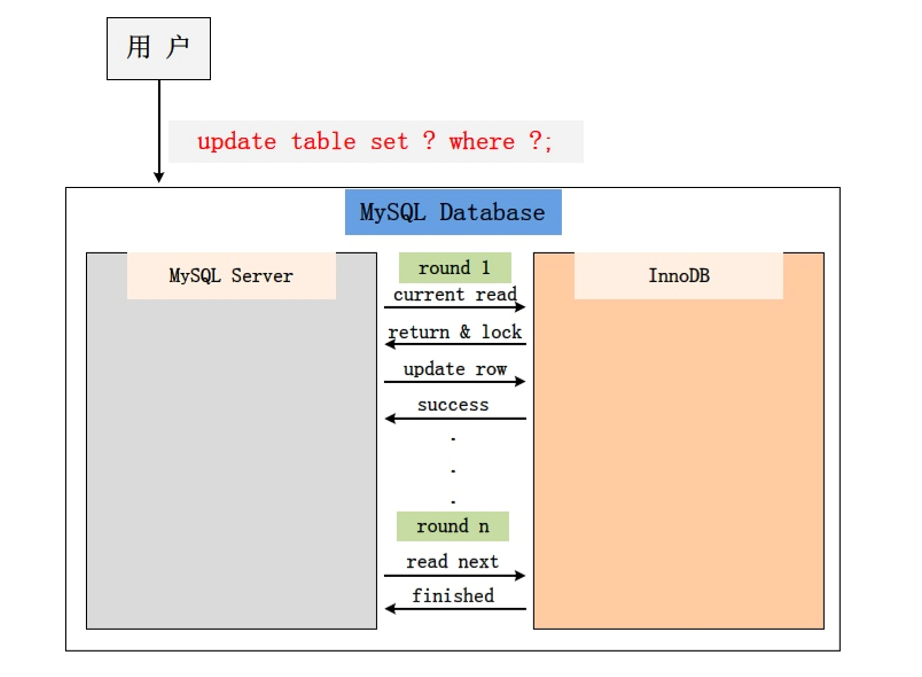
快照读忽略

针对当前读，RR隔离级别保证对读取到的记录加锁(记录锁),同时证对读取的范围加锁，新的满足条件的记录不能插入(间隙锁)，不存在幻读现象

##### Serializable

## 

#### 简单sql加锁流程

  
A:mysql 服务器查询满足where条件的记录

Binnodb 将返回第一条记录并加上锁

C:mysql server update这条记录

Dinnodb接收返回满足条件的一条记录

F：mysql server 更新

直到记录更新完毕

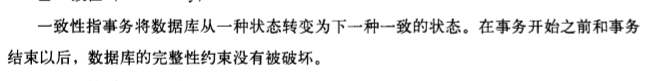
## 事务

### 特性

ACID

原子性，一致性，隔离性，持久性

整个数据库事务是不可分割的工作单位，事物的操作要么全部成功要么全部失败



一个事务的影响在其提交前对其他事务不可见

事务一旦提交，其结果是永久性的

### 事务的实现

隔离性由锁实现，原子性、一致性、持久性通过redo和undo实现

#### Redo

事务日志通过重做日志文件(redo)和InnoDB存储引擎的日志缓冲文件来实现

开始一个事务时，会记录该事务的一个LSN（日志序号列）

当事务执行时，回往innoDB存储引擎的日志缓冲文件里插入事务日志

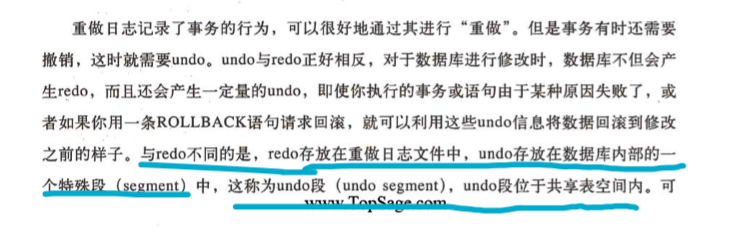
当事务提交时，必须将InnoDB存储引擎的日志缓冲写入磁盘

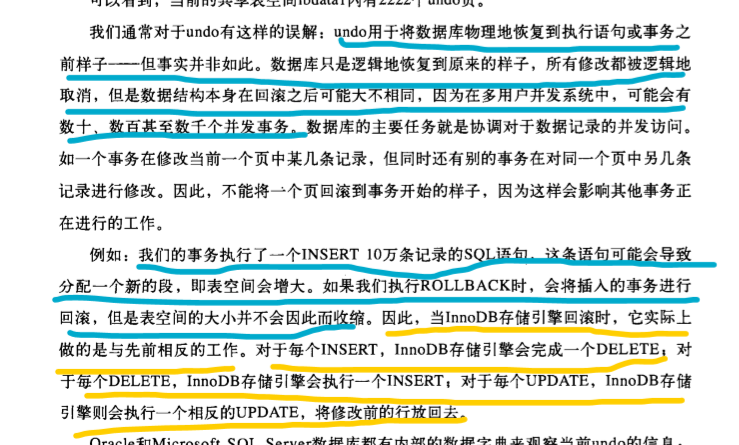
在写数据前需要先写日志，这种方式叫预写日志方式

InnoDB存储引擎通过预写日志的方式来保证事务的完整性。

#### Undo

事务有时需要撤销用undo日志





#### 分布式事务

##### 什么是分布式事务

分布式事务是指允许多个独立的事务资源参与一个全局的事务中，事务资源通常指关系型数据库系统，也可以是其他资源。

全局事务要求在其中所有参与的书屋要么都提交要么都回滚，

## 8.备份与恢复

### 8.1备份与恢复的概述

#### 备份方法分为：

热备 Hot Backup

冷备 Cold Backup

温备 Warm Backup

**热备**是指在数据库运行中直接备份，对正在运行的数据库没有任何影响。（在线备份）

**冷备**是指在数据库停止的情况下进行备份，最为简单，一般只需要拷贝相关的数据库文件即可。（离线备份）

**温备**是指在数据库运行时进行备份，但是会对当前数据库的操作有所影响。

#### 按照备份后的文件分为 逻辑备份 裸文件备份

**逻辑备份**是指备份后的文件是可读的，通常是文本文件，内容一般是SQL语句或者表内的实际内容。好处是可以看到导出文件的实际内容，一般用于数据库的升级、迁移等工作，但是恢复时间较长。

**裸文件备份**是指拷贝数据库的物理文件 ，数据库可以处于运行状态、或停止状态，这类备份恢复时间比逻辑备份时间短很多。

#### 按照备份数据库的内容可以分为：

**完全备份，增量备份，日志备份**

**完全备份**是指对数据库进行一个完整的备份

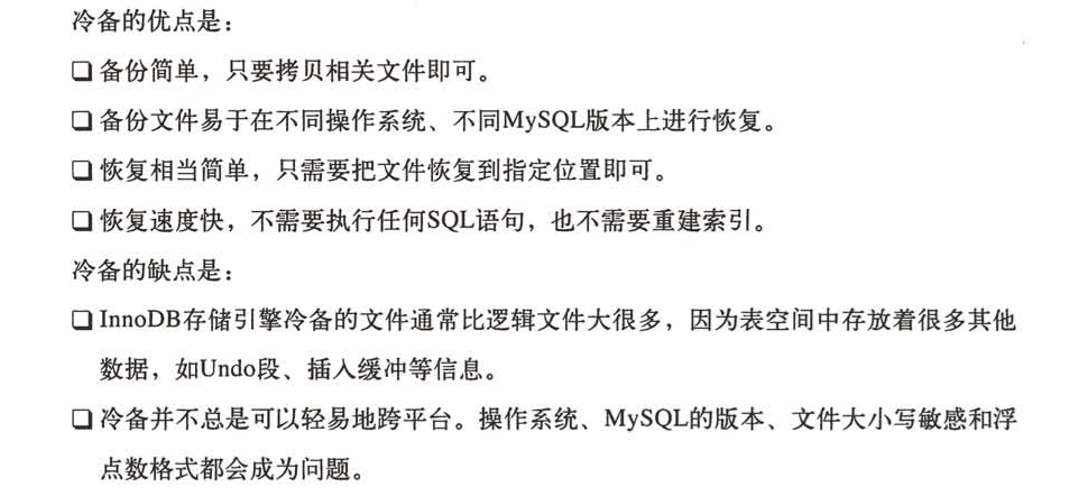
**增量备份**是指在上次的完全备份的基础上，对更新数据惊醒备份。

**日志备份**是指对mysql数据库二进制日志的备份，通过对一个完全备份二进制日志的重做来完成数据库的point-in-time的恢复工作。

Mysql数据库复制的原理就是异步实时进行二进制日志重做

### 8.2冷备

只需要备份mysql数据库的frm文件、共享表空间文件、独立表空间文件（\*.ibd）,重做日志文件 。定期备份mysqls的配置文件my.cnf，有利于恢复操作。



### 8.3逻辑备份

#### 8.3.1mysqldump

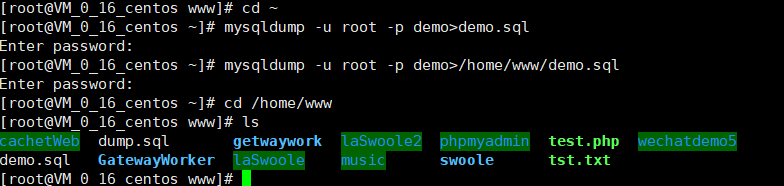
Mysqldump备份工具通常用来完成转存数据的备份以及不同数据库之间的移植，如 从低版本mysql数据库升级到高版本mysql数据库，或者从mysql数据库移植到Oracle和SQL server等

Mysqldump是mysql自带的备份工具，可以通过他的指令来进行备份，不需要进入 mysql –u root –p mysql界面

##### 备份指定的数据库到指定文件

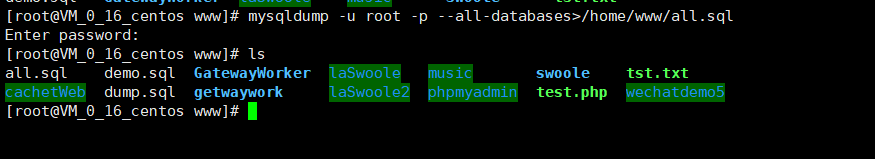
导出的数据包括数据下的表和表里面的内容

mysqldump –u root –p demo>/hemo/www/demo.sql



##### 备份全部数据库

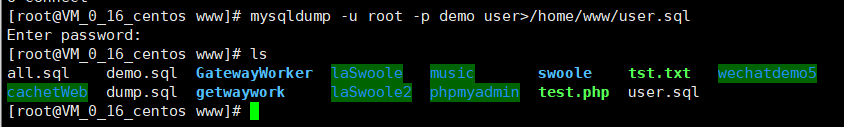
mysqldump -u root -p --all-databases>/home/www/all.sql



##### 备份指定数据库指定表

多个表逗号隔开

mysqldump -u root -p demo user>/home/www/user.sql



#### 8.3.2select … into outfile 导出一张表的数据

select \* into outfile '/var/lib/mysql-files/user.sql' from user;

mysql导出报错

Mysql数据库从文件导入或导出到文件，提示The MySQL server is running with the --secure-file-priv option so it cannot execute this statement

原因：

一些版本的mysql对通过文件导入导出作了限制，默认不允许，

查看配置，执行mysql命令

SHOW VARIABLES LIKE "secure\_file\_priv";

如果value值为null，则为禁止，如果有文件夹目录，则只允许改目录下文件（测试子目录也不行），如果为空，则不限制目录；

修改配置可修改mysql配置文件/etc/conf，查看是否有

secure\_file\_priv =

这样一行内容，如果没有，则手动添加，

secure\_file\_priv = /home

表示限制为/home文件夹

secure\_file\_priv =

表示不限制目录，等号一定要有，否则mysql无法启动

修改完配置文件后，重启mysql生效

#### 8.3.3逻辑备份的恢复

导入数据的时候没有demo数据库 报错 创建数据库 create database demo; use database;

进入mysql命令行界面

source /home/www/test.sql;

mysqldump -u root -p --all-databases>/home/www/all.sql

source /home/www/user.sql;

#### 8.3.4 LOAD DATE INFILE导入数据

通过mysqldump –tab或select into outfile导出的数据需要恢复时，可以通过 LOAD DATE INFILE进行导入

load data infile '/home/www/user.txt' into table user;

#### 8.3.5mysqlimport

Mysqlimport是mysql数据库提供的一个命令行程序，与 load data infile不同，mysqlimport可以导入多张表

Demo 指定数据库 user.txt test.txt为 user test表数据

mysqlimport -u root -p --use-threads=2 demo /home/www/user.txt /home/www/test.txt

### 8.4二进制日志备份与输入

MySQL 默认不启用二进制日志，

开启设置 在/etc/my.cnf 配置

log-bin

sync\_binlog=1

innodb\_support\_xa=1

第八章未看完

## 9.性能优化