InnoDB、MEMORY、BLACKHOLE、MyISAM、CSV、ARCHIVE、PERFORMANCE\_SCHEMA、FEDERATED

一：InnoDB引擎原理

1：mysql体系结构

Mysql的体系结构

Mysql由连接池组件、管理服务和工具组件、sql接口组件、查询分析器组件、优化组件、缓存组件、插件是存储引擎、物理文件

2:InnoDB引擎

InnoDB特点

Innodb存储引擎，特点支持外键、行锁、非锁定读(默认情况下读取不会产生锁)、mysql-4.1开始支持每个innodb引擎的表单独放到一个表空间里。Innodb通过MVCC来获取高并发性，并且实现sql标准的4中隔离级别，同时使用一种被称为next-key locking的策略来避免换读(phantom)现象。除此之外innodb引擎还提供了插入缓存(insert buffer),二次写(double write)，自适应哈希索引(adaptive hsah index),预读(read ahead)等高性能技术。

MyISAM索引与InnoDB索引相比较

\*myisam索引结构：myisam索引用B+tree来存储数据，myisam索引的指针指向的是键值的地址，地址存储的是数据

Innodb索引结构：用的B-tree来存储数据。

\*myisam支持全文索引(FULLTEXT),压缩索引，innodb不支持

\*innodb支持事务，myisam不支持

\*myisam顺序存储数据，索引叶子节点保存对应数据行地址，辅助索引跟主键索引相差无几；innodb主键节点同时保存数据行，其他辅助索引保存的是主键索引的值

\*myisan键值分离，索引载入内存(key\_buffer\_size)数据缓存依赖操作系统内；innodb键值一起保存，索引与数据一起载入innodb缓冲池

\*myisam主键(唯一)索引按升序来存储，innodb则不一定

\*myisam索引的基数值(cardinality,show index命令可以看见)是明确的,innodb则是估计值。这里涉及到信息统计的知识，myisam统计信息时保存磁盘中,在alter表或者analyze table操作更新此信息，而InnoDB则是在表第一次打开时候估计值保存在缓存区内。Myisam处理字符串索引是用增量保存的方式，如第一个索引是’preform’，第二个是’preformence’则是第二个保存’7,ance’这个明显的害处是缩短索引，但是缺陷就是不支持倒序提取索引，必须顺序遍历获取索引

Innodb引擎架构

InnoDB的多个内存块组成了内存池，负责如下工作：

维护所有进程/线程需要访问的多个内部数据结构；

缓存磁盘上的数据，方便快速的读取，并且在对磁盘文件的数据进行修改之前在这里缓存；

重做日志混村。后台线程的主要作用是负责刷新内存池中的数据，保证缓冲池中的内存缓存是最近的数据,此外,将已经修改的数据文件刷新到磁盘文件

1：后台线程

Innodb存储引擎后台有7个线程

4个IO线程(insert buffer thread,log thread,read thread,write thread),1个master thread 一个lock监控线程，一个错误监控线程

2：内存

InnoDB存储引擎内存由以下三个部分组成

\*缓冲池(buffer pool)

\*重做日志缓存(redo log buffer)

\*额外的内存池(additional memory pool)

可以使用show engine innodb status来查看innodb\_buffer\_pool的使用情况。

Innodb\_buffer\_pool\_size；具体看,缓冲池中的数据库类型有:索引页、数据库页、undo页、插入缓存页(insert buffer)、自适应hash(adaptive hashindex)、innodb存储的锁信息(lock info)、数据字典信息(data dictionary)。

\*innodb工作方式：将数据文件按页(每页16k)读入innodb的buffer pool,然后按最近最少使用算法(LRU)保留缓存数据,最后通过一定频率将脏页刷新到文件。

InnoDB关键特性,为inndob提高性能的技术

\*插入缓存

当一个表有非聚集索引时,对于非聚集索引的叶子节点的插入不是顺序的，这时候需要离散的访问非聚集索引页，性能就在这里降低了,这是由于b+树的原理所致的。插入缓存就是用来解决这个问题。

对于非聚集索引的插入和更新操作，不是每一次都直接插入索引页，二十先判断插入的非聚集索引页是否在缓存中,如果在就直接插入，如果不在就放入到一个插入换粗区中，好似欺骗数据库这个非聚集索引已经插入到叶子节点了.然后再以一定的频率插入缓存和非聚集索引页字节的合并操作。

插入缓存的使用需要满足以下两个条件(也就是非唯一的辅助索引):索引是辅助索引；索引不是唯一的；

\*两次写

两次写给innodb带来的是可靠性，主要用来解决部分写失败(partial page write).在应用重做日之前,我们需要一个页的副本。当写入失效发生是,先通过页的副业来还原该页，再进行重做,这就是doublewrite。

\*自适应哈希索引

由于innodb不支持hash索引，但是在某些情况下hash索引相率很高,于是出现了adaptive hash index功能,innodb存储引擎会监控对表上索引的查找,如果观察到建立hash索引可以提高性能的时候,则自动建立hash索引。

InnoDB使用MVCC特性

1：为什么使用MVCC :

对于事务型的存储引擎实现，仅仅依赖锁是不够的，还需要MVCC(Multiversion Concurrency Control)的帮助，可以简单的将MVCC理解成为一个row lock的一个变种，只是在必要的时候件、 行锁。

2：InnoDB的MVCC实现方式

详细：：<http://blog.sina.com.cn/s/blog_711b11fd0101bhks.html>

概述：

每个事务存储引擎的MVCC实现方式是不一样的，InnoDB的MVCC简单来说是通过给表添加两列隐藏列。其实是三个

\*一列(创建列)存储行的insert(如果行不存在)时间或者update(如果行已经存在)时间

\*一行存储行的删除时间，当然，这里的时间并非我们所说的时分秒，而是系统版本号(system version number)，列存储的SVN是事务开始时刻的SVN，每开始一个新的事务,svn号递增。

添加的三个列如下：

‘1DB\_TRX\_ID’:一个6byte的标识，每处理一个事务，其值自动+1,上述所说到的”创建时间”和”删除时间”记录的就是这个DB\_TRX\_ID的值，如insert、update、delete操作时,删除操作用1个bit表示。DB\_TRX\_ID是最重要的一个，可以通过语句”showengine innodb status”来查找，如下

‘2DB\_ROLL\_PTR：’大小是7byte,指向到rollback sgment(回滚段)的一条undo记录(update操作的是，记录update前的EOWz值

‘3DB\_ROW\_ID’:大小是7byte,该值随着新行插入单调的增加，当 由innodb自动产生聚焦索引时,聚焦索引包括这个DB\_ROW\_ID的值，不然的话聚焦索引中不包括这个值，这个用于索引当中。其次这里的不是真正的删除数据，而是标志出来的删除。正真意义的删除时早commit的时候。

1.VCC隔离级别的实现。

MVCC只有在隔离界别是READ COMMITED(Oracle默认)和REPEATABLEREAD(Mysql默认)两个隔离级别下工作

现在 在 REPEATABLE READ下的MVCC实现

\* SELECT

a.InnoDB查找SVN小于等于当前事务的SVN的行,如果小于，说明行之前就已经存在，如果等于，说明这行事物本省修改过的；

b行的删除时间列要么为空(说明改行为删除)要么删除时间列的SVN大于当前事务的SVN（表示行是在事物开始之后被删除 的）

只有记录满足以上两条，才会被select语句返回!

\*Insert

插入之后当以前的SVN号更新创建列

\*Delete

删除之后，当前SVN更新删除列

\*update

更新创建列为当前SVN,同时更新删除列为uodate之前创建的列创建的SVN值InnoDB 首先会从新复制一条记录写入到表中

这样的设计的有点是大部分的读操作的都不用加锁,使数据库操作简单，性能好,不足之处是增加了存储开销，需要额外的维护工作。

官方文档，关于MVCC的介绍是这样的：

InnoDB的MVCC是通过保存旧版本的修改信息来实现事物的并发控制和回滚。这些信息是以、一种数据结构的形式白存在表空间中，这种数据结构就是’rollback segment’回滚段可以提供两项功能:回滚，一致性读。

与MVCC相关的元素

DB\_TRX\_ID:占6字节，用来表示最近一次修改(insert/update)行事物的标识符，至于delete操作,在innodb看来也不过是一次uodate操作，更新行中一个特殊位将行表示为deleted.并非真正删除。

DB\_ROLL\_PTR：占用7个字节，滚回哦指针指向回滚段中对应回滚日志记录，如何update一行，那么改行旧信息就会被保存到指针所指向的回滚段中。

DB\_ROW\_ID：该值随行数增对而增加吗、，如果是cluster index，会包含这些开列，其他索引则不会有这列。

对于回滚段的管理

Insert logs ：当事物被提交的时，所保存的insert 回滚记录就可以被删除

Update logs ：这个日志不仅仅在rollback时需要，在一致性读也需要，所以不能随便删除,只有当数据库所使用的快照不涉及该日志记录，对应的回滚、日志才可以被删除

回滚段的管理指导：

经常将一些事物提交，包括哪些只发出一致性读的事物,我、否则innodb无法删除回滚段中的undo update logs，这些导致回滚段增长过快指导填满表空间。

在InnoDB中，发出Delete语句后，改行并不会立即被物理的从数据库中删除，只有当对应的行update undo logs被删除后，才会被物理的删除(purge)purge操作很快，时间等于delete所用时间

有个问题：在频繁的进行小规模的插入和删除操作，purge线程将会滞后。因为无法删除这些被标记为deleted的行,这会导致表越来越大，导致裁判瓶颈，性能下降。

解决办法：限制新行的操作，分配更多的资源purge线程。涉及参数unnodb\_max\_purge\_la当purge\_lags表示有经过delete或者update操作被标记为delete的行，没有被purge，如果该值超过innodb\_max\_purge\_lag值,则delete和insert,update操作将会被挂起。

MVCC实现原理以及实例化理解

下面用图像化形式表示MVCC如何处理select,insert,delete,update

有两个事务A、B

假设时间顺序ABCDqieDB\_TRX\_ID满足以下情况

A:DB\_TRX\_ID=2010

B:DB\_TRX\_ID=2011

C:DB\_TRX\_ID=2012

D:DB\_TRX\_ID=2013

注意:

1:B.DB\_TRX\_ID>A.DB\_TRX\_ID是因为DB\_TRX\_ID的值是系统版本号的值,系统版本号时候自动增加的，所以DB\_TRX\_ID也是自动增加的。但是会出现这种情况，假如A事务开始后B事务开始前有一个insert操作插入一行数据(没有bengin,comint)则B.DB\_TRX\_ID=A.DB\_TRX\_ID+1+1,并不符合不是说系统版本号增量为1，其实并不矛盾,其实每一条sql操作可以当做一个事务，因为autocommit=1.所以这个insert操作是一个事务,A事务之后新增加2个事务,所以是加2而不是1.

2：线面例化图只是笔者方便大家审计二设计的图片,红色代表隐藏两列

例化1：select

这是表test数据

Trx代表改行数据是那个事务创建

Create\_num是”创建时间”,也就是DB\_TRX\_ID值

Dele\_num是”删除时间”,空列代表没被任何事务标记为已”删除”,图中id为2的数据行的dele\_num=2012表示事务C”删除”了改行

B事务有select \* from test语句,按照MVCC原理，该语句相当于:select \* from test where creat\_num>=2011 and (dele\_num=NULL OR dele\_num>2011),所以返回数据是id为1、2行。

D事务select \* from test；则返回出id为1、3、4的行，因为2行被C事务删除了.

例化2:UPDATE

A事务一条语句”update from test set col=”winben” where col=”benwin”.

则先复制一条数据如蓝色框。Create\_num=DB——TRX\_ID(这里是2010),dele\_num=NULL,然后把旧行数据的设dele\_num=2010,等commi后删除就数据航

例化3:DELETE

删除就是设dele\_num=DB\_TRX\_ID

事务隔离级别和MVCC

1：READ NUCOMMITTED,不适用MVCC读,可以读到其他事务修改甚至未提交的

2：READ COMMITTED，其他事务对数据库的修改，只要已经提交，其修改的结果就是可见的，与这两个事务开始的先后顺序无关，不完全适用于MVCC读，

3：REPEATABLE READ 可以重复读，完全适用于MVCC,只能读取在开始之前已经提交的事务对数据库的修改，它在开始以后，所有其他事务对数据库的修改对它来说均不可见

4：SERIALIZBLE，完全不适用于MVCC,这样所有的query都会加锁，再它之后的事务都要等待 MVCC只工作在REPEATABLE READ和READ COMMITED隔离级别下

InnoDB的启动、关闭、恢复

Innodb\_fast\_shutdown影响innodb表关闭。该参数有0、1、2三个参数。

0：mysql关闭是，完成所有defull purge和merge insertbuffer操作

1：默认值 只将缓冲池内的一些脏页刷新至磁盘

2：将日志都写入日志文件不会有任何事务丢失但下次启动时徽进行recovery

Innodb\_force\_recovery影响整个innodb存储引擎的恢复情况，该值默认为0，当表示需要恢复时，需要执行所有的回复操作，当不能进行有效恢复时，如数据也发生了corruption,mysql数据库可能宕机，并把错误写入日志中

日志文件

1：错误日志

错误日志对mysql的启动，运行，关闭过程进行了记录。出现Mysql不能正常启动时，第一个必须查找的文件就是错误日志文件，使用show variable like ‘log\_error’来定位文件。

2：慢查询日志

慢查询能为sql语句的优化带来很好的囊祝。设定一个阈值，将运行时间超过该值的所有sql语句都记录到慢查询日志文件中。用参数long\_query\_time来设置。另一个参数log\_queries\_not\_using\_indexes，若运行的sql语句没有使用缩影，则这条sql语句会被记录下来。

3：查询日志

查询日志记录了所有对mysql请求的信息,不论这些请求是否得到正确的执行。默认文件名为:主机名.log

二进制日志：

二进制记录了对数据库执行更改的所有操作，但是不包括seletc和show操作，还包括了执行时间和更改操作时间，可用来恢复某些数据同时也可以用来复制同步远程数据库，将binlog\_format设置为row，可以支持事务隔离级别为READ COMMITTED，以获得更好的并发性。在使用MIXED格式下，mysql采用STATEMENT格式进行二进制日志文件的记录,但是有一些亲狂下会使用ROW格式，可能的情况如下：

1：表的存储引擎为NDB，这个时候DML操作都会以ROW格式记录

2：使用了uuid(),user(),current\_user(),found\_rows(),row\_count()等不确定函数。

3：使用insert delay语句

4：使用了用户定于的函数(UDF)

5:使用了临时表(temporary table)

注意:针对系统库mysql里面的表发生的处理规则如下：

1：如果采用insert,update,delete直接操作表，则日志根据binlog\_format设定的格式记录。

2：如果使用grant，revoke,set password等DCL语句，那么无论如何都会使用SBR模式记录。

3：blockhole引擎不支持row格式,ndb引擎不支持sattement格式

InnoDB引擎文件

表空间文件：

默认表空间文件为ibdata1文件innodb\_data\_file\_path存储数据，innodb\_file\_per\_table可以按表分别产生一个表空间.db文件。但仅存该表的数据缩影和插入缓存等信息，其他信息如undo信息，系统事务信息,double write buffer等还是存放在默认表空间(ibdata1或表空间组)里。

重做日志文件：

Redo log 是在实例或者介质失败的时候，用来保证数据完整性，每个innodb 存储引擎至少有一个重做日志组，每个重做日志组文件组下至少又2个重做日志文件，如默认的ib\_logfile0,ib\_logfile1，为了得到更高的可靠性，你可以设置多个重做镜像日志组。

因为重做日志且先被写到日志缓冲中，然后根绝一定条件刷星到磁盘重做日志文件中，与redolog相关的就是innodb\_flush\_log-at\_trx\_commit的值，对innodb的性能影响很大。他有0,1,2,三个值，0代表提交事务是，并不同步写redo——log,而是等masterthreas每秒写。1代表commit的时候就将redo-log写入磁盘，2代表commit的时候将redo log缓存异步的写入磁盘。

InnoDB索引

InnoDB使用何种索引-----使用约束索引

当数据库一条记录里包含多个字段时，一棵B+树就只能存储主键，如果检索的是非主键字段，则主键索引失去作用，又变成顺序查找了，这是应该在第二个检索的裂伤建立第二套索引。这个索引有独立的B+树来组织

有两种产检的方法可以解决多个B+树访问同一套表数据的问题：

一种叫做聚簇索引，一种叫做非聚簇索引。这两个民资虽然都叫做索引，但这并不是一种单独的索引类型，而是一种数据存储方式，对于聚簇索引存储来说，行数据和主键B+树存储在一起，辅助键B+树只存储辅助键和主键，主键和非主键B+树几乎是两种类型的树。对于非聚簇索引存储来说，主键B+树在叶子节点存储指向真正数据行的指针，而非主键。

InnoDB使用的是聚簇索引，将主键组织到一棵B+树中，而行数据就绪存储在叶子节点上，若使用’where id=14’这样的条件查找主键，则按照B+树的检索算法即可查找对应的叶子节点，之后获得到行数据，如对Name列进行条件搜索，则需要两个步骤：第一步在辅助索引B+树检索操作，最终到达叶子节点即可获取整行数据。

Myism使用的是非聚簇索引，非聚簇索引的两课B+树看上去没什么不同，节点的结构完全一致只是存储的内容不同而已，主键索引B+树的节点存储了主键，索引键索引B+树存储了辅助键。表数据存储在独立的地方，这两棵B+树的叶子节点都是用一个地址指向真正的表数据，对于表数据来说，这两个键没有任何差别。由于索引树是独立的，通过辅助键检索无需访问主键的索引树。

为了更形象说明这两种索引的区别，我们遐想一个表如下图存储了4行数据。其中ID作为主索引，Name作为辅助索引。图示清晰的显示了聚簇索引和非聚簇索引的差异。

ID-----Name------Job

我们重点关注聚簇索引，看上去聚簇索引的效率明显要低于非聚簇索引，因为每次使用辅助索引检索都要经过连词B+\_树查询，这不是多次一举吗？聚簇索引的优势在哪？

1：由于行数据和叶子节点存储在一起，这样主键和行数据是一起被载入内存的，找到叶子节点就可以立刻将行数据返回了，如果按照主键ID来组织数据，获得数据更快。

2：辅助索引使用主键作为“指针”而不是使用地址作为指针的好处是，减少了当出现异动或者数据页分裂时辅助索引的维护工作，使用抓紧啊值当做指针会让辅助索引占用更多的空间，换来的好处是Innodb在一定是无须跟新辅助索引中这个指针。U、也就是行的位置(实现通过16k的page来定位，后面会设计)会随着数据库里面的数据的修改而发生变化(前面的B+树节点分裂以及page的分裂)，使用聚簇索引就可以保证不管这个抓紧啊B+树的节点如何变化，辅助索引树都不受影响

InnoDB的索引类别

默认情况下Memory使用存储hsah索引，但也支持b+tree索引。Hash索引只用于=或者⬄的等式比较，不能用来家督order by 操作，只能通过关键字搜索一行

Innodb只支持b+树索引,进一步分为clustered index 与 secondaryindex.在一次查询中，只能使用一个索引

Innodb是索引组织表：

InnoDB结构上索引分为两类

\*clustered index 主键索引：

叶节点保存着整行的数据。如果，定义了primary key ，则cluster index就是primary key的索引：如果没有定义primary mysql 会选中第一行第一个仅有的not null列的unique索引作为主键，并把此索引当做clustered index使用；如果没找到这样的列,innodb会创建一个6字节的Rowld作为主键，所以每张表有且仅有一个clustered index。

\*Secondary index辅助索引

叶节点不包括行的全部数据，包含键值以外还包括一个bookmark，可以告诉innodb到什么地方可以找到相对应的完整行数据，还保存了主键的键值。Secondary index包含主键，但不包含完整的行数据，所以InnoDB总是会先从secondary index的叶子节点判断是否能得到所需的数据，如

Create table t(a int , b varchar(20),primarykey(a),key(b);

Explain select\* from t;

会发现mysql选择了索引b，而不是a

\*复合索引

复合索引是在多列(>=2)上建立的索引，又叫多列索引或联合索引。Innodb中的复合索引也是b+tree结构。索引的数据包含多列(col1,col2,col3…..)，在索引中依次按照col1,col2,col3排序，如(1,2),(1,3),(2,0)…..

使用复合索引要充分利用最左前缀原则，顾名思义，就是最左右线。如创建索引ind\_col1\_col2（col1,col2）,那么在查询where col1=xxx and col2=xx或者where col1=xxx 都可以zouind\_col1\_col2索引

在创建多列索引时，要根据业务需求，where自居中使用最频繁且过滤效果好的一列放在最左边。

索引操作

可以通过DML语句操作innodb索引。因为innodb是索引组织的表，对索引的操作会造成锁表，先生成一张临时表，将数据从原始表中写道临时表，再将原始表示删除，最后将临时表给名成原始表表名，因增加，删除，修改字段会对主索引产生影响，所以也会锁表。对secondary index从innodb plugin开始，支持快速索引创建的方法，在创建的过程中不需要重建表，所以速度会很快，同事引擎会在表上加s所，在创建过程中只能进行读操作。

索引设计原则

1：搜索的索引列，不一定是所要选择的列。也就是说，最适合索引的列是出现在where字句中的列，或者链接字句中指定的列。而不是出现在select关键字后的选择列表中列。

2：使用唯一索引。考虑某列的分布，索引的列的级数越大，索引的效果越好。例如，对性别M/F做索引没有多大作用。

3：使用短索引。如果是对字符串进行索引，如果有可能应该指定前缀长度。

4：利用最做前缀。尽量将使用频繁且过滤效果好的字段放在左边

5：不要过度索引

6：innodb默认会按照一定的顺序保存数据，如果明确定义了主键，则按照主键顺序保存。如果没有主键，但有唯一索引，就按照唯一索引的顺序保存。如果有几个列都是唯一的，都可以作为主键的时候，为了提高查询效率，应选择常用访问的列作为主键。灵位，innodb的secondary index都会保存主键的键值，所以主键要尽可能选择较短的数据类型。可以看出，应当尽量避免对主键的修改。经过对dba的测试，保证主键的递增可以提高插入性能。

Mysql如何使用索引

1：对于创建的多列索引，只要查询的条件中用到了最左边的列，索引一般就会被使用。

2：对于使用like的查询，后面如果是常量并且只有%号不在第一个字符，索引才可能内使用。

3：如果对大文本进行搜索，应该使用全文索引，而不是使用like’%...%’但不幸的是innodb不支持全文索引。

4：如果列名是索引，使用index\_column is null将使用索引。Oracle是不行的。

5：如果mysql估计使用索引比全表索引更慢，最不会使用索引。

6：如果使用memory/head表并且where 条件中不适用”=”进行索引列，那么不会用到索引。Head表只有在”=”进行索引咧，那么不会用到索引。Head表只有在’=’的时候才会使用索引。

7：用or分割开的条件，如果or前的条件中的列有索引，二后面列中没有索引，那么涉及到的索引都不会被用到。

8：不是多列索引的第一部分不会走索引

9：以%开始的like不会走索引

10：如果列是字符串，那么一应要在where条件中把字符串常亮值用来引导起来，否则不能走索引。因为，mysql默认把输入的常量值进行转换后才能进行检索。

11：经过普通运算或函数运算后的索引字段不能使用索引

12：不等于操作不能使用索引,<>,not in等

13：Order by优化：某些情况下，mysql可以使用一个索引满足order by而不是需要额外的排序。Where条件与order by 使用相同的索引，并且order by 顺序和索引顺序相同，并且order by 的字段都是升序或者都是降序。

SELECT \* FROM t1 ORDER BY key\_part1,key\_part2,... ;

SELECT \* FROM t1 WHERE key\_part1=1 ORDER BY key\_part1 DESC, key\_part2

DESC;

SELECT \* FROM t1 ORDER BY key\_part1 DESC, key\_part2 DESC;

但是以下情况不使用索引：

SELECT \* FROM t1 ORDER BY key\_part1 DESC, key\_part2 ASC ；

--order by 的字段混合 ASC 和 DESC

SELECT \* FROM t1 WHERE key2=constant ORDER BY key1 ；

-- 用于查询行的关键字与 ORDER BY 中所使用的不相同

SELECT \* FROM t1 ORDER BY key1, key2 ；

-- 对不同的关键字使用 ORDER BY

可以使用explain查看sql的执行计划。

InnoDB的存储

表空间是逻辑存放所有数据的地方，默认情况下会共享一个表空间—ibdata1，但如果把innodb\_file\_per\_table=ON后每张表可以单独放到一个表空间内，但还是有很多数据保存在共享的表ibdata1中，如undo信息等

表空间有各种段(segemtn)组成，常见的段有数据段、索引段等。InnoDB是索引组织的，数据段就是clustered index的叶子节点。需要注意的是，不是每个对象都有段。

区（extend）是由64个连续的页组成，每个页（page）固定为16KB,所以每个区总共为1M,页是innodb最小的磁盘管理单位。

InnoDB是按行进行存放的，每个区最少可以保存2条记录，否则就成链式结构了。每行数据除了字定义列外，还会增加事务id和回滚指针列。如果没有定义primary key也没有not null的unique,则会增加6字节的ROwld列作为主键。

InnoDB表的限制

一个表不能包含超过1000列。

内部最大键长度是3500字节，但mysql自己限制这个到1024字节。

除了VARCHAR,BLOB和TEXT列，最大行长度稍微小于数据页的一般，即，最大行长度大约8000字节，ONGBLOB和LONGTEXT列必须小于4GB,总的行长度，页包括BLOB和TEXT列，必须小于4GB,InnoDB在行中存储VRRCHAR，BLOB或TEXT列的前768字节，余下的存储的分散的页中。

虽然InnoDB内部地支持行尺寸大于65535，您不能定义一个包含VARCHAR列的，合并尺寸大于65535的行。

mysql> CREATE TABLE t (a VARCHAR(8000), b VARCHAR(10000),

· -> c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),

· -> f VARCHAR(10000), g VARCHAR(10000));

· ERROR 1118 (42000): Row size too large. The maximum row size for the

· used table type, not counting BLOBs, is 65535. You have to change some

· columns to TEXT or BLOBs

在一些更老的操作系统上，数据文件必须小于2GB.

InnoDB日志文件的合并尺寸必须小于4GB

最小的表空间的尺寸是10MB,最大的表空间尺寸是4000000000个数据库页（64TB）.这也是一个表的最大尺寸。InnoDB表不支持FULLTEXT索引。

二：MEMORY引擎原理（内存的存储引擎）

使用Mysql Memory存储引擎的出发点是速度。使用的逻辑存储介质是系统内存