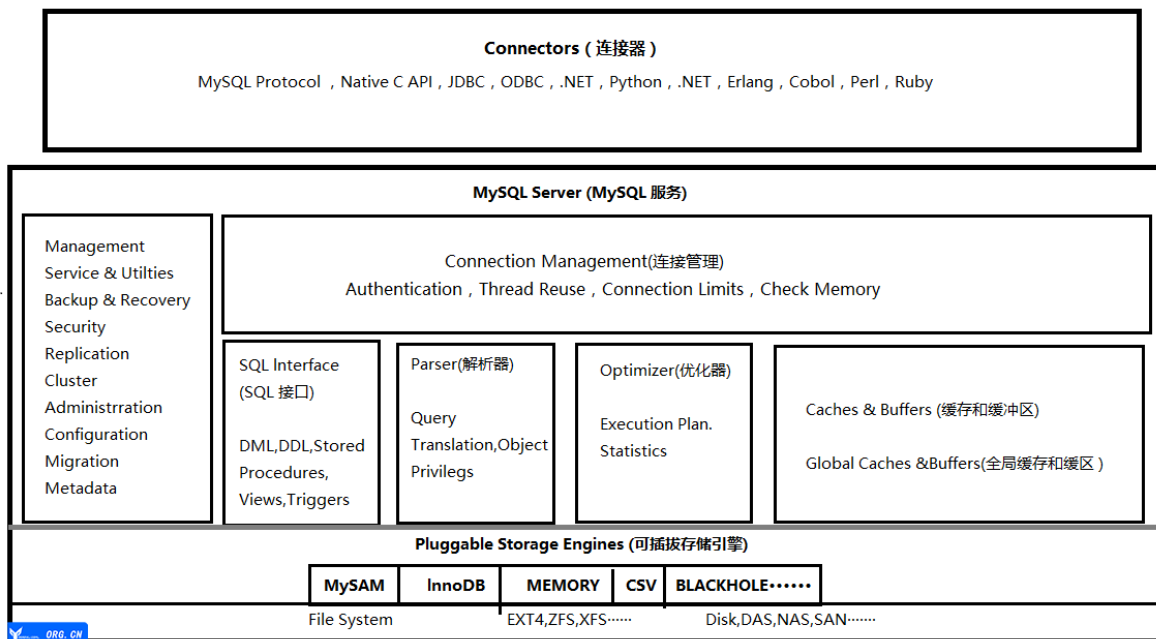


存储引擎

一，MySQL体系结构



connectors属于应用层；

而在MySQL Server以下则属于服务层；

Connection Management 为SQL的连接层共包括；权限认证，线程复用，连接限制，内存检查

SQL Interface (SQL接口)、Parser (解析)、Optimizer (优化)、Caches & Buffers (内存控制) 四个分别为SQL层

pluggable Storage Engines：MyISMA、InnoDB、MEMORY、CSV、BLACKHOLE为SQL的储存引擎，决定了数据的存取方式

二，MySQL所支持的引擎及常见的储存引擎

使用 show -engines \G; 来查看当前数据库所支持引擎

Engine: ， 名称

Support: 当前服务是否支持

Comment: 备注

Transactions: 事务

XA: 是否支持分布式

Savepoints: 是否支持保存点

常见引擎及相关介绍：

MEMORY： 储存引擎

特点在于 .frm文件存储表的结构信息，数据存放在内存中，没有表数据文件，重启后数据会丢失。使用表级锁，表的最大大受到max_heap_table_size的参数限制

用于适用主内容变化不频繁的表，或者作为中间的查找表

演示案例：create table t_meml(id int) engine=memory;

创建一个MEMORY的指定引擎数据表。

使用 insert into t_meml values(1);

insert into t_meml values(2);

增加两行数据。

并使用select * from t_meml; 查询

数据可查，但没有数据文件，且重启后数据会丢失。但表不会丢失。

使用一下命令以增加存储：

set max_heap_table_size=1024*1024 ;

create table t_mem_1m(v1 varchar(10000)) engine=memory ;

set max_heap_table_size=1024*1024 ;

create table t_mem_2m(v1 varchar(10000)) engine=memory ;

增加t_mem_1m 和 t_mem_2m 其存储大小为 1024

使用 show table status like 't_mem%' \G ; 来查看详细信息

也可以使用 show table status like 't_mem%' ;查看

CSV：存储引擎

特点在于.csv文件存储表内容，.csm文件存储表的元数据如表的状态和数据量，.frm文件存储表的结构信息，所有列非空not null，不支持索引分区。

适用于数据存储为CSV文件格式，不用进行转换

演示案例

```
CREATE TABLE t_csv1 (id int not null,c1 VARCHAR(10) not null) ENGINE = csv;
```

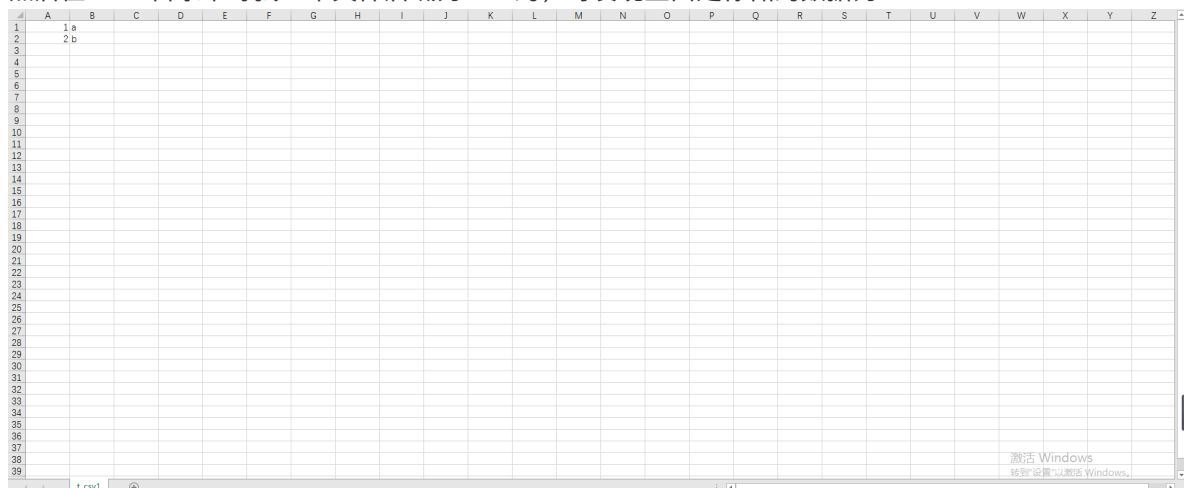
指定t_csv1表为csv 引擎

insert into t_csv1 values(1,'a');

insert into t_csv1 values(2,'b');

之后使用 select * from t_csv1 ; 查看表。

然后在/data目录下寻找一个文件后缀为.CSV的，可发现里面是存储的数据为 a b



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	1 a																									
2	2 b																									
3																										
4																										
5																										
6																										
7																										
8																										
9																										
10																										
11																										
12																										
13																										
14																										
15																										
16																										
17																										
18																										
19																										
20																										
21																										
22																										
23																										
24																										
25																										
26																										
27																										
28																										
29																										
30																										
31																										
32																										
33																										
34																										
35																										
36																										
37																										
38																										
39																										

ARCHIVE：存储引擎

ARCHIVE存储引擎的特点在于 .frm文件存储表的结构信息，arz文件存储表的数据。支持Insert、replace和celect操作，但是不支持Updata和delete，往archive表插入的数据会经过压缩，压缩比非常高，存储空间大概是innodb的10~15分之1

适用于数据归档，采集等场景

演示案例：

```
create table t_ar(id int) engine=archive ;  
创建一张表名为 t_ar，指定引擎为 archive
```

```
insert into t_ar values(1);  
增加一行数据为 1
```

```
insert into t_ar values(2);  
增加一行数据为 2
```

当你看你的终端总共有3个 OK 时，使用 `select * from t_ar;` 查看表

当你输入 `delete from t_ar;` 的时候，则会弹出
ERROR 1031 (HY000): Table storage engine for 't_ar' doesn't have this option
的警告信息
错误1031(HY000):“t_ar”的表存储引擎没有此选项

所以Archive引擎是不允许删除和修改的。

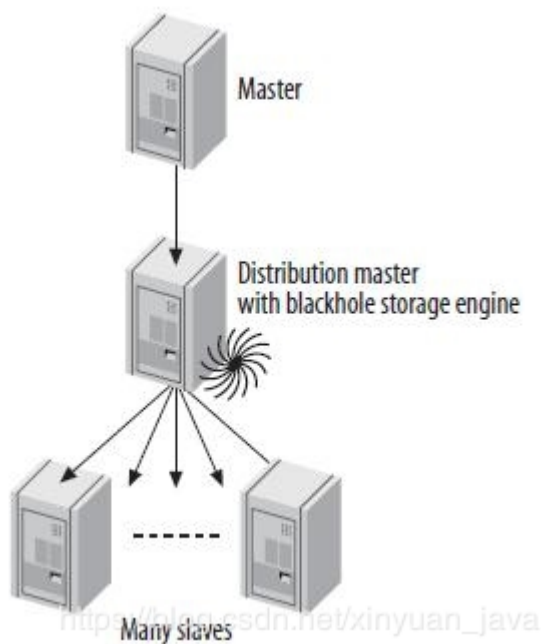
BLACKHOLE：存储引擎

BLACKHOLE的特点在于只有表结构文件.frm，但没有表数据文件
任何写入到此引擎的数据均会被丢弃掉,但是会在binlog日志中记录

可适用于做日志记录或同步归档的中继存储

应用场景

在配置一台一主多从的服务器中，多个服务器会在主服务器上分别开启自己相对应线程，执行 binlog dump命令，而且多个此类进程并不是共享的，是为了避免多个服务器同时请求而造成主机资源耗尽，可以单独的建立一个伪的从服务器或者叫分发服务器



可以从上图可以理解到，一台主服务器的如果要分发1w的数据是非常的吃力的，如果使用BLACKHOLE黑洞引擎则会减少很多的存储，因为他是不浪费太多的存储空间，因为都在binlog日志当中。

演示实例：

```
create table t_bhl (i int,cl char(10)) engine = BLACKHOLE ;  
创建一个表名为 t_bhl，指定引擎为BLACKHOLE
```

```
insert into t_bhl values(1,'record one'),(2,'record two');  
为表增加一行数据
```

当你使用 `select * from t_bhl;` 是查询不到任何结果的，

MRG_MYISAM：存储引擎

MRG_MYISAM存储引擎表本身保存数据，而是只起到汇总的作用，将一组结构相同的MyISAM表逻辑上聚合到一起。由.frm文件储存表的结构信息，.mgr不保存数据，保存的是数据的来源。

适用于单表过大，水平分表等。

演示案例

```
create table t_mys1 (id int not null auto_increment primary key,cl varchar(20)) ENGINE=MyISAM ;
建立一张表为t_mys1，指定引擎为myISAM；
```

```
create table t_mys2 (id int not null auto_increment primary key,cl varchar(20)) ENGINE=MyISAM ;
建立一张表为t_mys2，指定引擎为myISAM；
```

```
insert into t_mys1(cl) values('this'),('is'),('mysql');
增加一行数据为CL，其数据内容为this is mysql
```

```
insert into t_mys1(cl) values('this'),('is'),('mysql2');
增加一行数据为CL，其数据内容为this is mysql
```

这时可以使用 `select * from t_mys1` 查看创建表的已有数据。

```
create table t_mer1(id int not null auto_increment primary key,cl
varchar(20))ENGINE=MRG_MyISAM UNION=(t_mys1,t_mys2);
```

使用以上语句对 t_mys1和t_mys2进行合并内容。

如果想 t_mysql2 插入数据则使用

```
insert into t_mys2(cl) values ('this'),('is'),('mer1');, 此时在t_mys1 也有数据。
```

FEDERATED：存储引擎

FEDERATED的特点在于跨服务访问数据，.frm文件储存表的结构信息。

但是你想跨服务访问的话则需要用到FEDERATED引擎，跨服务访问指的是两个系统之间数据库的访问。

就比如你在MySQL中的zsdk库中查询user库，则属于夸库查询

案例演示

在另一台MySQL中设置

```
1.CREATE TABLE t_fd (id int(11) NOT NULL,name varchar(30) NOT NULL,age int(11) NOT
NULL,PRIMARY KEY (id))ENGINE=InnoDB CHARSET=utf8;
```

定义一个表名为t_fd的数据库，引擎为InnoDB，字符编码为utf-8

```
2.insert into t_fd (id,name,age)values(0,'zhangsan',20);
```

在t_fd表中插入数据为id,name,age, zhangsan

在你的机器上设置

```
3.CREATE TABLE t_fd (id int(11) NOT NULL,name
varchar(30) NOT NULL,age int(11) NOT NULL,PRIMARY KEY (id) )ENGINE=FEDERATED
CONNECTION='mysql://root:wotamazhenshuai@192.168.78.131:3306/mysite/t_fd';
首先你需要创建和另一台mysql中一样的表，然后连接到对方服务器,其账号是root，密码是
wotamazhenshuai，连接地址为192.168.78.131/mysite/t_fd
```

MyISAM 引擎

MyISAM引擎主要包裹.frm储存表对象结构，.MYD数据文件，存储数据，.MYI索引文件，用于存储表的索引信息

MyISAM引擎的特点

不支持事务，（事务是指逻辑上的一组操作，组成这组操作的各个单元，只有成功和失败一个结果）

表级锁：锁定的实现成本很小，但是降低了其并发性

读写互相阻塞：写入时会引起阻塞读取，还会在读取的时候阻塞写入，但是如果是只读的方式不会阻塞。

组缓存索引：MyISAM可以通过Key_buffer_size缓存索引，提高了访问性能，减少磁盘IO,但是缓存区只会缓存索引，不会缓存数据

不支持外键约束

MyISAM特点在于适用于非事务性系统，读写并发访问较少的业务系统

非事务性系统

对事务性表和非事务性表 放在一个事务中，插入后不进行commit 而是进行 rollback操作（注：此时根据提示warning，非事务性表无法回滚

MyISAM引擎格式

静态：表中不包函类型的列(varchar/varbinary/blob/text)

动态：包含变长字符类型的列(varchar/varbinary/blob/text)

压缩：通过myisampack创建

```
create table t_mysam(id int,cl char(10)) ENGINE=MyISAM;
```

建立一个库名为 t_mysam的，且指定为MyISAM引擎，静态

```
create table t_mysam2(id int,cl varchar(10)) ENGINE=MyISAM;
```

建立一个库名为 t_mysam2的，且指定为MyISAM引擎，动态

此时会在/data/下目录生成三个文件，frm,myd,myi等。

静态格式的特点以及压缩格式的特点

查询的速度较快，崩溃后容易重新组建，因为记录保存的位置是固定的，通常比动态格式的表占用更多的磁盘空间

静态与动态的存储相比较

就比如我们将 a和b来做存储，静态的虽然只占两个字符，但是后面的需要用 000000000 补齐，而动态的则不需要0000 补齐，动态只占两个字符。

所以动态表存储的是字符实际需要的空间，但是崩溃的时候也不仅更难恢复

而压缩格式的特点往往是只。读，不支持添加任何的修改。

InnoDB存储引擎

InnoDB引擎设计的时候遵循了ACID模型，支持事务，支持行级锁，以提供过用户并发时的读写性能，Innodb引擎表组织数据是按照逐渐聚簇，支持外键约束，拥有服务彭桂中的恢复能力，简称自我恢复，可以极大地保证数据安全，innodb拥有自己的缓存池而且还是独立的，常用的数据及索引都缓存当中。

InnoDB事务

ACID模型

Atomic（原子性）所有的语句作为了一个单元全部成功执行或全部取消，不能出现总监阶段。

就比如你在找女朋友当中，一个貌美如花一个漂亮贤惠而你只可以得到其中一个妹子，你如果多选了你违反规则，除非你是渣男。

Consistent（一致性）

直属局食物不能破幻关系数据的完整性以及业务逻辑上的一致性。

Isolated (隔离性)

事物之间不会产生互相的影响。(事务是在连接的时候发生的,两者不会互相影响,就比如你连接a1,隔壁老王连接a2,是不会互相影响的)

Durable (持久性)

事物成功完成之后,所做的所有更改都会准确的记录在数据库当中,且做的更改不会丢失(包括数据库崩溃的时候也是不会丢失的)

而这些所做的都是为了ACID模型中的一致性,数据库事务不能破坏关系数据的完整性以及业务逻辑上的一致性。

一句话,保证完整性。

事务概述

事务是数据库区别于系统文件的重要特性之一,事务是指逻辑上的一组操作,组成这组操作的各个单元,只能有一个结果,同时,事务也有严格的定义,他必须同时满足ACID

事务控制

字啊默认的情况下,连接到MySQL服务的客户端处于自动提交模式,也就是说每天DML执行就是提交,如果启用事务支持,有两种方法

禁用事务自动提交

MySQL中默认提交功能由系统变量autocommit控制,该变量值为0或者off即可禁用自动提交,僵尸舞的提交(commit)回滚(rollback)控制权交由前端用户控制

案列演示

查看变量autocommit

```
show variables like 'autocommit';
```

```
set autocommit=off;
```

设置autocommit禁止自动提交

```
create table t_innodb1(id int not null auto_increment primary key,cl varchar(20))
```

```
auto_increment=1;
```

插入一张名为 t_innodb1的表,且没有指定innodb引擎。

```
show create table t_innodb1;
```

来查看查看,发现

```
PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |  
+-----+-----+  
1 row in set (0.01 sec)
```

默认指定了innodb引擎

```
insert into t_innodb1(cl) values('a');
```

提交一个id, cl, 1, a数据

```
select * from t_innodb1;
```

可发现数据提交成功

如果想取消操作则可以使用回滚

rollback;则取消了表中的所有操作

显示声明事务

执行DML语句前,先通过start transaction或begin语句启动一个事务执行SQL语句后,就可以通过commit或rollback语句来控制事务的提交或回滚。

start transaction ;

开启事务

delete from t_innodb1 ;

执行事务

select * from t_innodb1 ;

查看事务

commit ;

提交事务

经过以上三联，t_innodb1已经没有数据了。

使用后三联提交数据

三连之前使用命令

set autocommit=on ;

show variables like 'autocommit' ;

如果autocommit 是ON状态则可以三联

start transaction ;

insert into t_innodb1(cl) values('a');

insert into t_innodb1(cl) values('b');

insert into t_innodb1(cl) values('c');

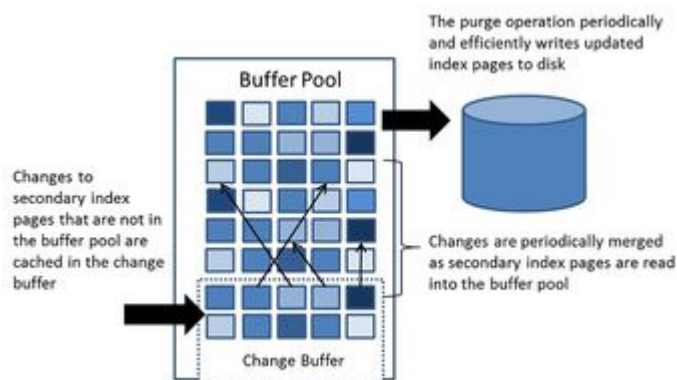
之后使用 select * from t_innodb1 ;查看，则为Empty set (0.00 sec)，如果不为Empty set (0.00 sec) 需要使用

rollback ; 回滚

则数据为 Empty set (0.00 sec)

使用show engines \G: 查看支持引擎中发现，只有InnoDB支持事务。

InnoDB引擎内存结构



Buffer Pool

InnoDB专用缓存，用来缓存表对象的数据和索引信息的，大小由innodb_buffer_pool_size变量指定，默认为128MB，独立的数据库服务器中，该缓存大小可以设置为物理内存为 80 %

show variables like 'innodb%buffer%';

show engine innodb status \G;

使用以上两个命令都可以查看内存结构

Change Buffer

Change buffer的主要目的是将二级索引的操作（IDU）缓存下来，而不是直接读入索引页进行更新，在择机将change buffer中的记录合并到真正的二级索引中，以及减少二级索引的随机IO

innodb_change_buffer_max_size：表示change buffer在buffer pool中的最大占比，默认为25百分之，最大为百分之五十

在MySQL5.5之前的版本中，由于只支持缓存insert操作，最初叫做insert buffer，只是后的版本中还吃了更多的操作类型缓存，才改名为change buffer。

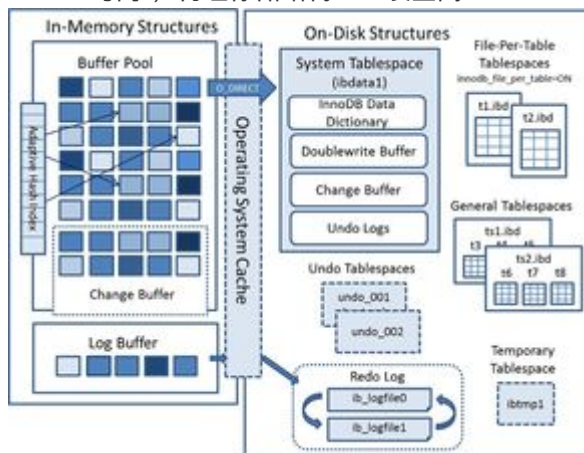
总结下来就是Change Buffer就是增加二级索引，增加随机IO的~

Log Buffer

存储要写入日志文件的数据存储区域，Log Buffer 的大小由innodb_log_buffer_size变量定义，默认大小为16MB.Log Buffer的内容会定期刷到磁盘上。

Log Buffer缓存的数据主要会定期写入在Redo Log当中

InnoDB引擎，物理存储结构——表空间



主要包含：

系统表空间

默认情况下InnoDB引擎只对应一个表空间，即系统表空间，所有InnoDB引擎表的数据索引都在该表的空间当中，注意的是保存数据和索引，表对象结构信息人保存在 .frm文件中

InnoDB系统表空间对应那些物理数据文件，通过系统变量 innodb_data_file_path指定，其语法为：
innodb_data_file_path = file_name:file_size[:autoextend[:max:max_file_size]]

file_name：指定文件名

file_size：指定文件初始化大小

autoextend：指定文件是否可扩展，可选参数

:max:max_file_size：指定该数据文件最大可占用空间，可选参数

MySQL5.7中，autoextend默认以此扩展为64M空间，可通过innodb_autoextend_increment 系统变量指定

在默认的情况下，ibdata1:12M:autoextend 保存在 E:\mysql-5.7.26-winx64\data\ 目录下

独立表空间

在默认的情况下，InnoDB引擎的表和索引都保存在系统表空间对应的数据文件中，数据文件越大，管理成本越高，系统表空间的数据文件扩展后无法会所，及时表被DROP或TRUNCATE,已分配的空间任然是相对于InnoDB数据可用，不能被其他操作系统在分配给其他文件使用

如果出现这种情况可以考虑应用InnoDB数据存储的另一项设定，InnoDB将其定义为多重表空间（multiple tablespaces），就是每个表对象拥有一个独享的.ibd为扩展名的数据文件，这个数据文件就是一个独立的表空间。

是否启用多重表空间是由系统变量 innodb_file_per_table 控制

如果你开启了一个独立表空间，则会生成单独的.ibd 文件，如果没有则整合到ibdata1

独立表空间与优点：

各个表对象的数据独立存储至不同的文件，可以灵活的分散 I/O ,执行备份及恢复操作

当执行TRUNCATE/DROP删除表对象时，空间即时释放回操作系统层。

案例演示

use zsdk ;

进入一个库

drop table t_mer1 ;

释放 t_mer1

cishi t_mer1已被释放回系统层

Redo日志

redo日志仅针对InnoDB引擎，MySQL数据库的其他引擎是用不到的，默认情况下InnoDB引擎会创建两组大小均为5MB的日志文件，fenbiemingnmingwei ib_logfile0和ib_logfile1，日志文件保存在datadir变量指定的目录下，不过可以通过InnoDB的专用参数修改日志路径，日志文件组数量及日志文件的大小。

innodb_log_file_size：大小

innodb_log_files_in_group：组数

innodb_log_group_home_dir：路径

redo在事务中的应用

Redo的作用；

redo来实现事务持久性，redo对于AC也有响应的作用，可参考ACID模型

持久性相关组件

重做日志缓存（redo log buffer）是容易丢失的。

从左日志文件（redo log file）是持久的

持久性的原理

当食物提交（Commit）时，会刷新当前事务的redo buffer到重做日志文件中持续持久化，待事务的commit完成CIA算弯沉，会将此日志打赏commit标记，还会顺便将一部分redo buffer中没有提交的日志刷新到redo日志文件中

在Log Buffer写入的时候，及时你一直写入，虽然传输到了Redo Log，但不会打上提交完成的标签

Undo日志

对于实务操作来说，又提交就必然会有回滚，提交就是确定保存写入到数据，那么回滚就是代表着两部的操作，首先撤销刚刚做的修改，而后在数据恢复至修复前的状态，那么数据一经洗写入，最简单恢复修改前的状态就是在修改前将旧数据保存下载，保存下的这部分数据就是 UNDO日志，粗存在系统分配好的回滚端中。

总结就是 undo日志是保存旧数据的一个日志

create table t_idb1(id int not null auto_increment primary key,cl varchar(20))engine=innodb

auto_increment=1;

创建一张表，指定了InnoDB引擎

insert into t_idb1(cl) values('a');

插入一个数据a

select * from t_idb1 ;

查看当前第二列第二行数据为 b

```
rollback;
```

回滚，此时做了两个操作，第一个是将b删除，第二个是把a恢复

使用

```
select * from t_idb1 ;
```

可发现已经将b调换成a了、

回滚端中的UNDO日志共分为两种，

insert UNDO：尽在事务回滚时需要，事务提交后被删除

update UNDO：用于构造一致性读，当没有任何事务需要使用到行记录的之前版本时才会被遗弃

因为InnoDB的回滚端，一致性的这类特征，事务的信息不要长期存放，否则会因为回滚段过大导致沾满整个系统表空间，拖累InnoDB引擎运行（如果没有输入rollback 或者说 commit时，会一直记录，所以会被沾满）

insert UNDO

开始

```
start transaction ;
```

插入一个信息为b的插入到t_idb1

```
insert into t_idb1(cl) values('b');
```

提交

```
commit ;
```

开始

```
start transaction ;
```

查看 t_idb1 表信息

```
select * from t_idb1
```

为t_idb1增加'c'字段;

```
insert into t_idb1(cl) values('c');
```

查询 t_idb1

```
select * from t_idb1 ;
```

回滚

```
rollback ;
```

查看from t_idb1表信息

```
select * from t_idb1 ;
```

此时发现刚刚添加的c字段被滚出去了

UNDO日志存储结构

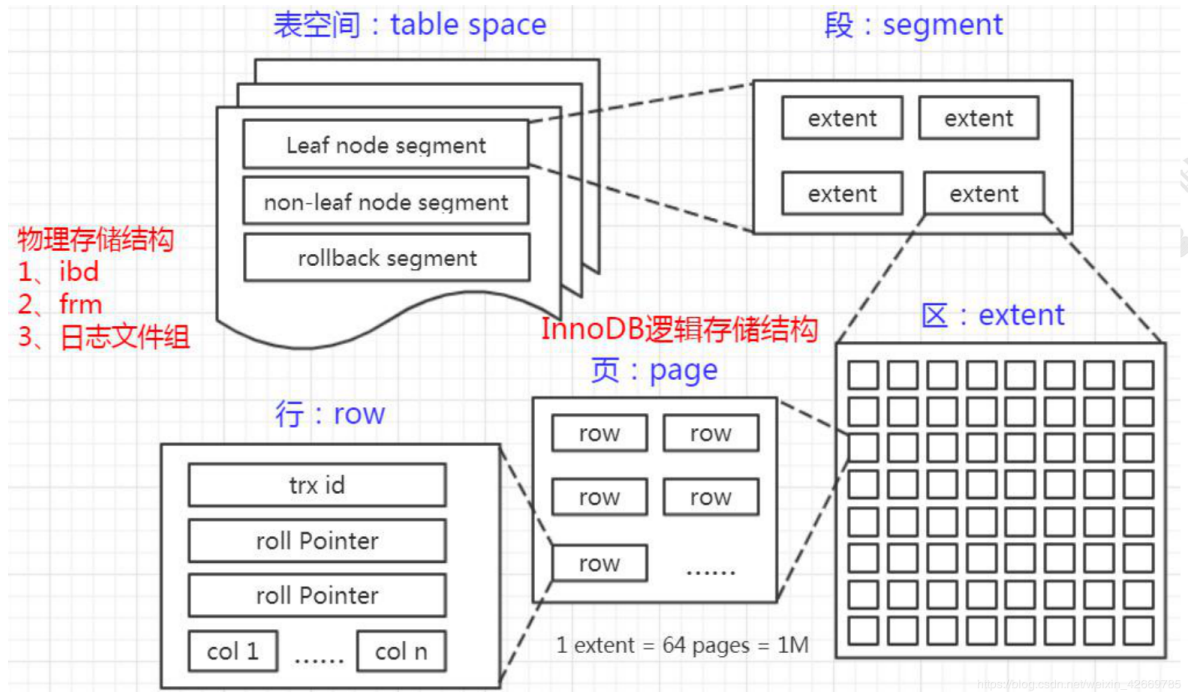
默认情况下，UNDO日志存储在系统表空间，InnoDB引擎中的UNDO日志可以设置单独的空间。

innodb_undo_directory：指定单独存放undo表空间目录，默认为datadir（MySQL5.7版本中，只支持初始化设置不可中途开启）

innodb_rollback_segments：回滚端的数量，至少等于35.默认128

innodb_undo_tablespaces：指定单独存放的undo表空间个数，必须在MySQL初始化阶段设置，初始化完成后就不能再修改。

InnoDB逻辑存储结构



主要分为 tablespace、segment、row、page、extent。

主要组成顺序为 row、page、extent、segment、tablespace

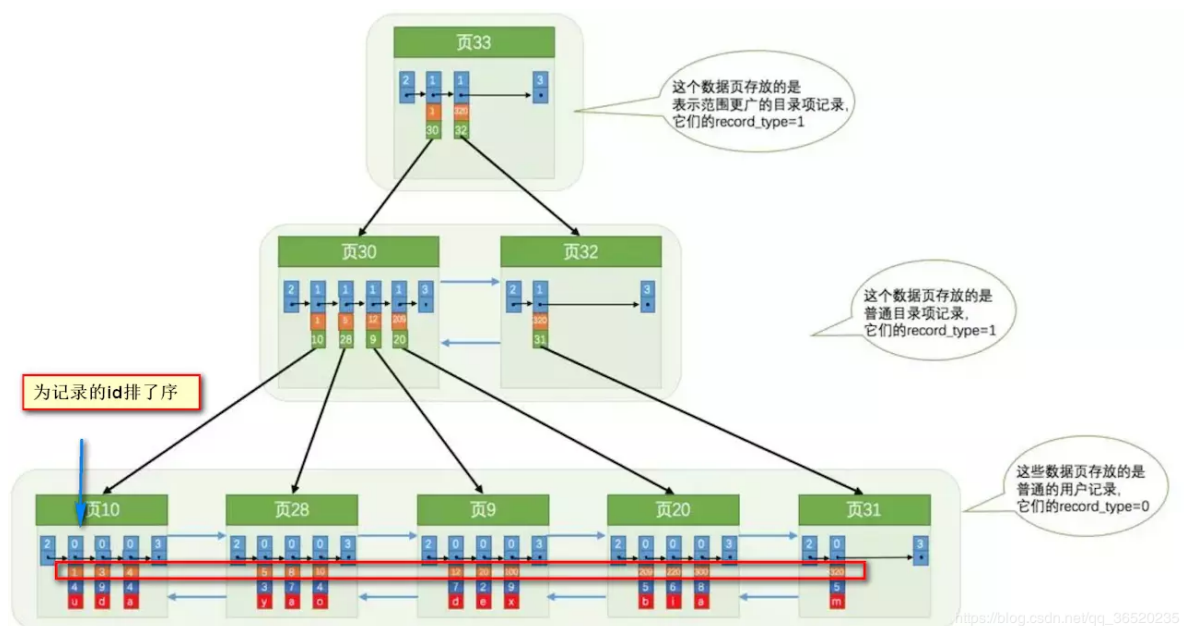
页：是InnoDB中的最小的管理单位，同一个MySQL服务，所有表空间都拥有相同的页大小，迷人情况是16kb，可以再初始化数据库是通过innodb_page_size变量进行配置，可选值有4\8\16KB

区：每个区固定大小为1mb，由16个16kb的页组成

段:Segment有无数个Extent组成，段有 数据段，索引段，回滚段等。

表空间：逻辑存储单元最高粒度

如果需要扩展的表空间，InnoDB第一次分配给32个pages之后每次扩展会分配一个完整的Extent给Segment，最大能够同时向Segment中砸呢家4个Extent，以确保数据的连续性

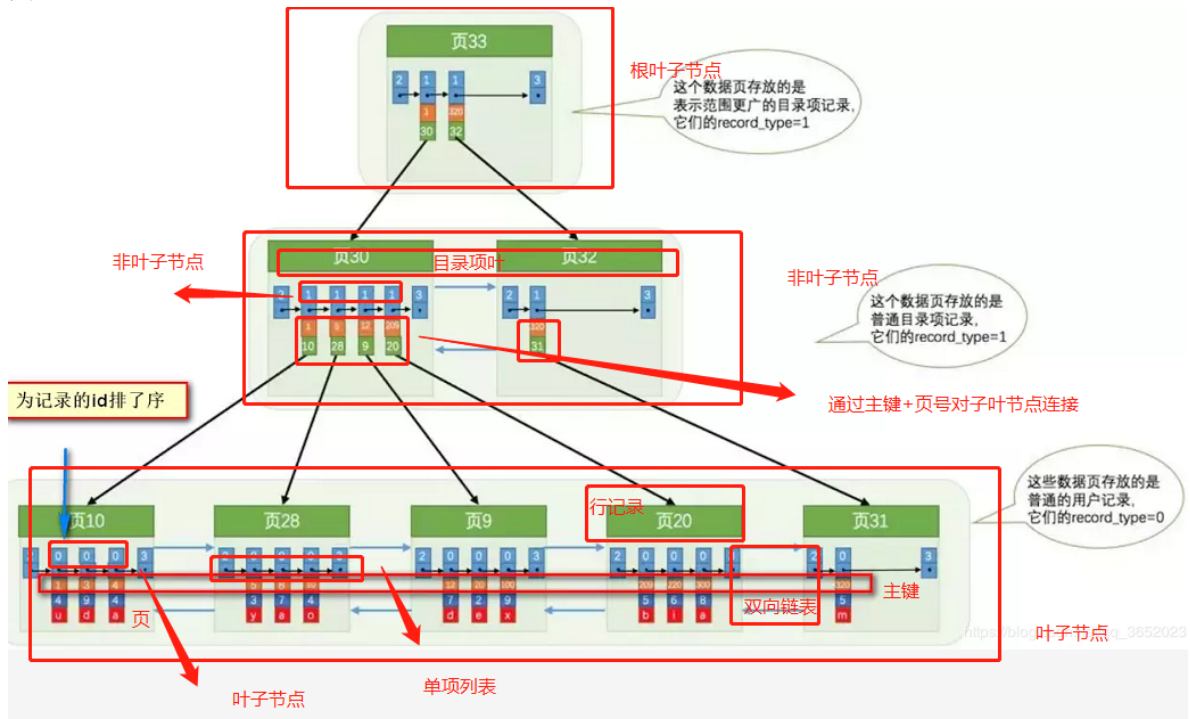


聚簇索引

主键+页号

双向链表

单向链表
目录项表
非叶子及单
叶子节点
主键（非空和唯一）
行记录
页号
页



使用记录主键值的大小进行记录和页的排序，这包括三个方面的含义

页内的记录是按照主键的大小顺序排成的一个单向链表

各个存放用户记录的也是根据也中记录的逐渐打消顺序排成的一个双链表

各个存放目录项的页也是根据也中的记录的逐渐打消顺序排成的一个双向链表

B+树的叶子结点存储是完整的用户记录

聚簇索引

我们吧具有这两种特征的B+数为聚簇索引，用户记录都存放在这个居村索引的叶子节点处。

这种聚簇索引并不需要我们在MySQL语句中显示的去创建，InnoDB存储引擎会自动为我们创造举出索引，在InnoDB存储引擎中，举出索引就是为了数据的存储方式（书哟偶用户记录都纯纯姿叶子节点上）也就是所谓的索引即 数据。

二级索引

上边介绍的聚簇索引只能在搜索条件是主键值的时候才能发挥作用，因为B+数中的数据都是按照主键进行排序的，如果我们想以别的列作为搜索条件时，如何便利？

可以多见几个B+树，不同的B+树种的数据采用不同的排序规则，比如用c2列值的大小作为数据写中的排序规则，在建议可B+树效果图如下：

二级索引和聚簇索引的不同之处

使用记录c2列的大小进行记录和页的排序，这包括三个方面的含义；

也的记录是按照c2列值的大小顺序排成的一个单向链表

各个存放用户记录的也是根据也中的c2列大小排序成一个上香链表

各个存放目录项的也是根据也中的记录c2列大小顺序你排成的一个双向链表
B+树的叶子节点存储的并不是一个完整的用户及诶按，而是一个c2列和主键
目录项记录中不再是主键+页号的，而是变成了c2lie+页号