17.MySQL存储引擎–体系结构√

1.存储引擎简介

1.0 存储引擎介绍

> 相当于Linux 中的文件系统，和“磁盘”。负责IO的控制（磁盘资源、内存资源、对象资源）。

1.1 InnoDB 存储引擎核心特性介绍

> MVCC： 多版本并发控制
> 聚簇索引： 用来组织存储数据和优化查询
> 支持事务： 数据最终一致提供保证
> 支持行级锁： 并发控制
> 外键： 多表之间的数据一致一致性
> 多缓冲区支持
> 自适应Hash索引： AHI
> 复制中支持高级特性。
> 备份恢复： 支持热备。
> 自动故障恢复：CR Crash Recovery
> 双写机制： DWB Double Write Buffer

2.存储引擎管理

2.1 查询支持的存储引擎

Bash | Copy

```
查询所有可用的存储引擎种类
mysql> show engines;
查询、设置默认存储引擎
mysql> select @@default_storage_engine;
+--------------------------+
| @@default_storage_engine |
+--------------------------+
| InnoDB                   |
+--------------------------+
1 row in set (0.00 sec)

vim /etc/my.cnf
default_storage_engine=InnoDB
```

2.2 查看某张表的存储引擎

mysql> show create table xta;
查询系统中所有业务表的存储引擎信息
mysql> select
table_schema,
table_name ,
engine
from information_schema.tables
where table_schema not in
('sys','mysql','information_schema','performance_schema');
巡检需求： 将业务表中所有非InnoDB查询出来
mysql> select table_schema,table_name,engine from information_schema.tables
where table_schema not in
('mysql','sys',

2.3 创建表设定存储引擎

Bash | Copy

```
mysql> create table xxx (id int) engine=innodb charset=utf8mb4;
```

2.4 修改已有表的存储引擎

Bash | Copy

```
mysql> alter table world.xxx engine=innodb;
```

2.5 将所有的非InnoDB引擎的表查询出来，批量修改为InnoDB

客户案例：将历史遗留下的非InnoDB表进行处理
1.查询所有非InnoDB表
mysql> select table_schema,table_name ,engine
from information_schema.tables
where
table_schema not in
('sys','mysql','information_schema','performance_schema')
and engine !='innodb';
2. 备份所有非InnoDB表
select concat("mysqldump -uroot -p123 ",table_schema," ",table_name,"
>/tmp/",table_schema,"_",table_name,".sql")
from information_schema.tables

where
table_schema not in
('sys','mysql','information_schema','performance_schema')
and engine !='innodb';
3. 修改存储引擎
mysql> select concat("alter table ",table_schema,".",table_name,"
engine=innodb;") from information_schema.tables where table_schema
not in ('sys','mysql','information_schema','performance_schema') and
engine !='innodb' into outfile '/tmp/a.sql';

mysql> source /tmp/a.sql

2.6 查询碎片情况

查询碎片：

information_schema.tables; ---> DATA_FREE

alter table world.xxx engine=innodb;

analyze table world.city;

或者导出导入一下。

3.InnoDB体系结构---线程和内存结构详解

3.1 线程结构

1.Master Thread

a. 控制刷新脏页到磁盘（CKPT）

b. 控制日志缓冲刷新到磁盘（log buffer ---> redo）

c. undo页回收

d. 合并插入缓冲(change buffer)

e. 控制IO刷新数量

说明：

参数innodb_io_capacity表示每秒刷新脏页的数量，默认为200。

innodb_max_dirty_pages_pct设置出发刷盘的脏页百分比，即当脏页占到缓冲区数据达到这个百分

比时，就会刷新innodb_io_capacity个脏页到磁盘。

参数innodb_adaptive_flushing = ON（自适应地刷新），该值影响每秒刷新脏页的数量。原来的

刷新规则是：脏页在缓冲池所占的比例小于innodb_max_dirty_pages_pct时，不刷新脏页；大于

> innodb_max_dirty_pages_pct时，刷新100个脏页。
>
> 随着innodb_adaptive_flushing参数的引入，InnoDB存储引擎会通过一个名为
>
> buf_flush_get_desired_flush_rate的函数来判断需要刷新脏页最合适的数量。粗略地翻阅源代
>
> 码后发现buf_flush_get_desired_flush_rate通过判断产生重做日志（redo log）的速度来决
>
> 定最合适的刷新脏页数量。因此，当脏页的比例小于innodb_max_dirty_pages_pct时，也会刷新一
>
> 定量的脏页。

2.IO Thread

> 在InnoDB存储引擎中大量使用Async IO来处理写IO请求,IO Thread的工作主要是负责这些IO请求的回调处理。
> 写线程和读线程分别由innodb_write_threads和innodb_read_threads参数控制，默认都为4。

3.Purge Thread

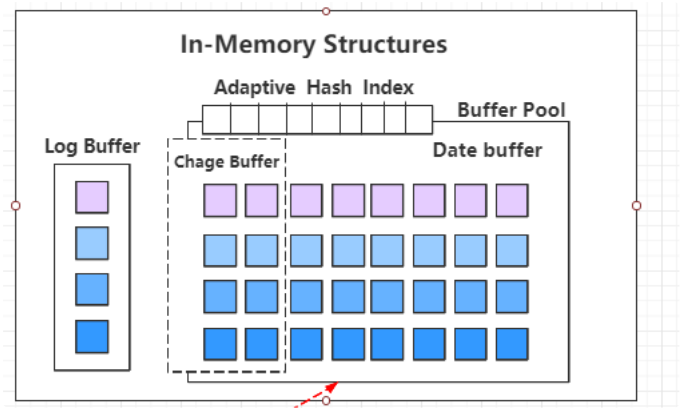
> 事务在提交之前，通过undolog(回滚日志)记录事务开始之前的状态，当事务被提交后，undolog便不再
>
> 需要，因此需要Purge Thread线程来回收已经使用并分配的undo页。可以在配置文件中添加
>
> innodb_purge_threads=1来开启独立的Purge Thread，等号后边控制该线程数量，默认为4个。

4.Page Cleaner Thread

> InnoDB 1.2.X版本以上引入，脏页刷新，减轻master的工作，提高性能。
>
> 查看方法：
>
> 客户端版本：
>
> mysql –V
>
> server 版本：
>
> select @@version;
>
> engine 版本：
>
> SELECT * FROM information_schema.plugins;
>
> SELECT @@innodb_version;

MySQL版本	Innodb引擎版本
5.1.x	1.0.x版本(官方称为InnoDB Plugin)
5.5.x	1.1.x版本
5.6.x	1.2.x版本

3.2 内存结构



1.InnoDB Buffer Pool(IBP)

作用：

用来缓冲、缓存，MySQL的数据页（data page ）和索引页、UNDO。MySQL中最大的、最重要的

内存区域。

管理：

查询：

> select @@innodb_buffer_pool_size;

> select @@innodb_buffer_pool_instances;

默认大小： 128M

生产建议： 物理内存的：50-75%。

在线设置：

> mysql> set global innodb_buffer_pool_size=268435456;

> 重新登录mysql生效。

> 永久设置：

> vim /etc/my.cnf

> #添加参数

> innodb_buffer_pool_size=256M

> show global status like '%innodb%wait%';
> 产生不够用的情景有哪些?

> 1. 设置太小。

> 2. 大事务。

> 3. chpt触发不及时。

> 4. IO比较慢。

> 5. 查询语句优化的不好。

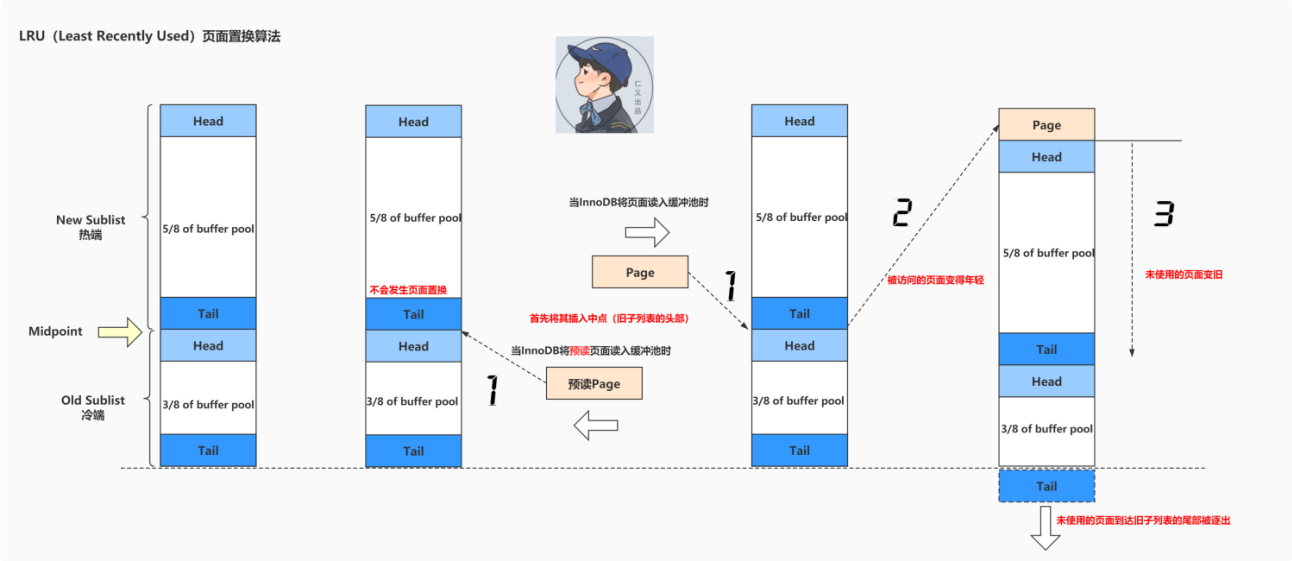
2.LRU算法

> 1. 采用LRU：最近最少使用原则。

> 2. LRU链表：“热端”（5/8）+“冷端”（3/8）

> 3. 新入内存的page，会被放在冷端的头部。

4. 如果page被访问，会被调取到热端的



头部。

3.Change buffer

比如insert，update，delete 数据。

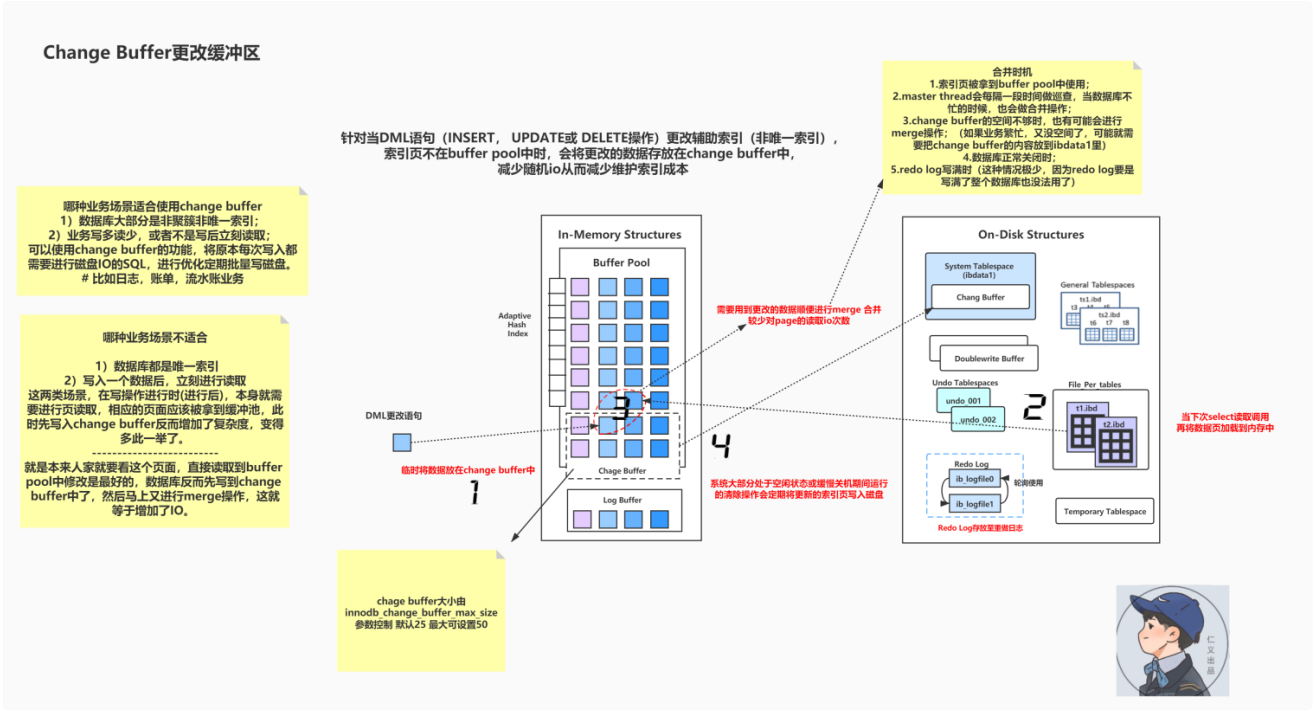
对于聚簇索引会立即更新。

对于不在buffer pool的辅助索引需要更新，不是实时更新的。

在InnoDB 内存结构中，加入了insert buffer，现在版本叫change buffer。

Change buffer 功能是临时缓冲辅助索引需要的数据更新。

当我们需要查询新insert 的数据，会在内存中进行merge(合并)操作，此时辅助索引就是最新的。



4.AHI 自适应hash索引

> MySQL的InnoDB引擎，能够创建只有Btree。

> AHI作用：

> 自动评估"热"的内存索引page,生成HASH索引表。

> 帮助InnoDB快速读取索引页。加快索引读取的效果。

> 相当与索引的索引。

> metux 闩锁争用。

5.Log Buffer（redo buffer）

> 作用： 用来缓冲 redo log日志信息。

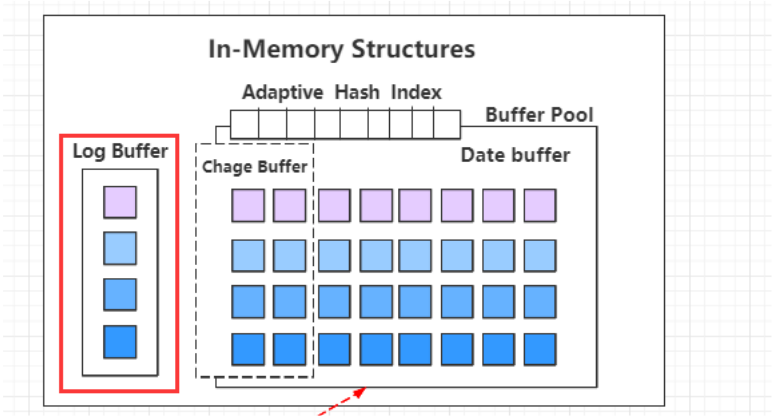
> 管理 ：

> 查询：

> mysql> select @@innodb_log_buffer_size;

> 默认大小：16M

> 生产建议：和innodb_log_file_size有关，1-N倍
>
> 设置方式 ：
>
> vim /etc/my.cnf
>
> innodb_log_buffer_size=33554432
>
> 重启生效：
>
> [root@db01 data]# /etc/init.d/mysqld restart
>
> show global status like '%innodb%log%';

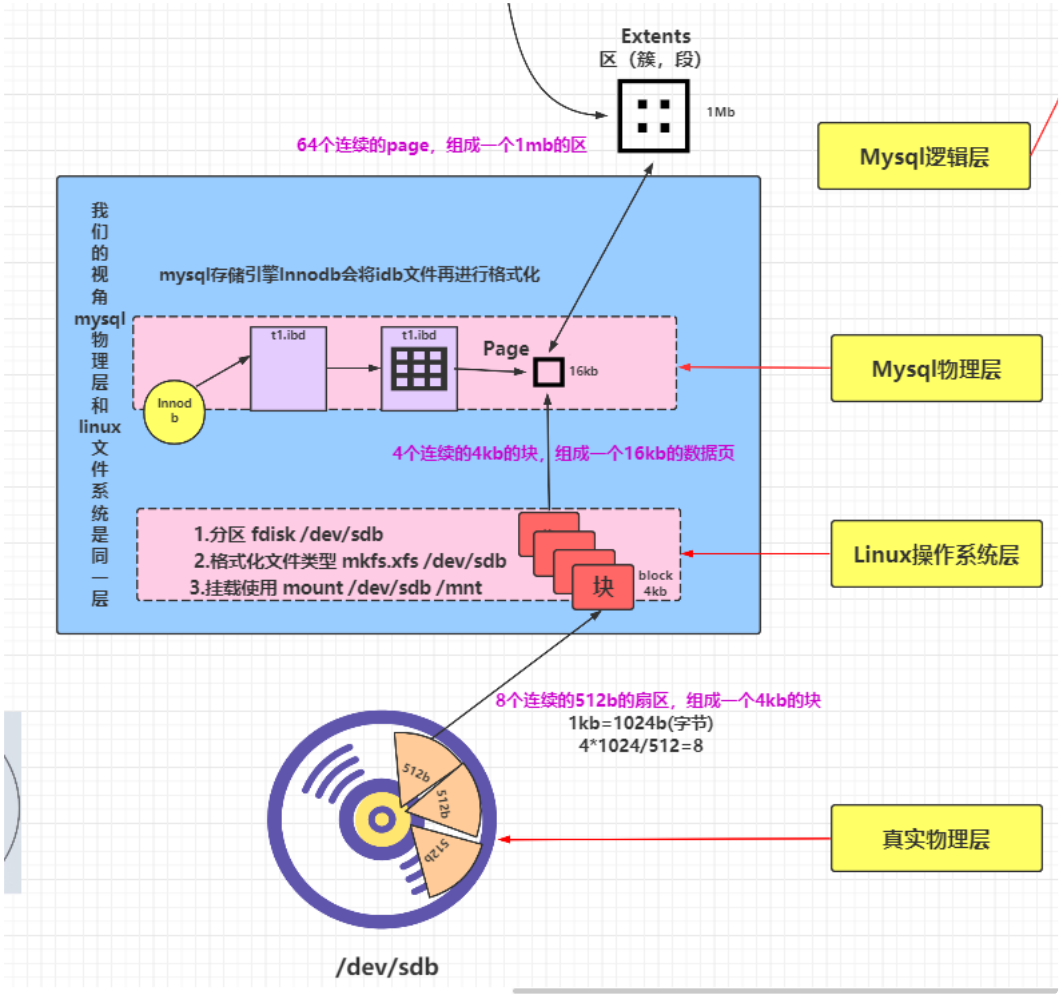


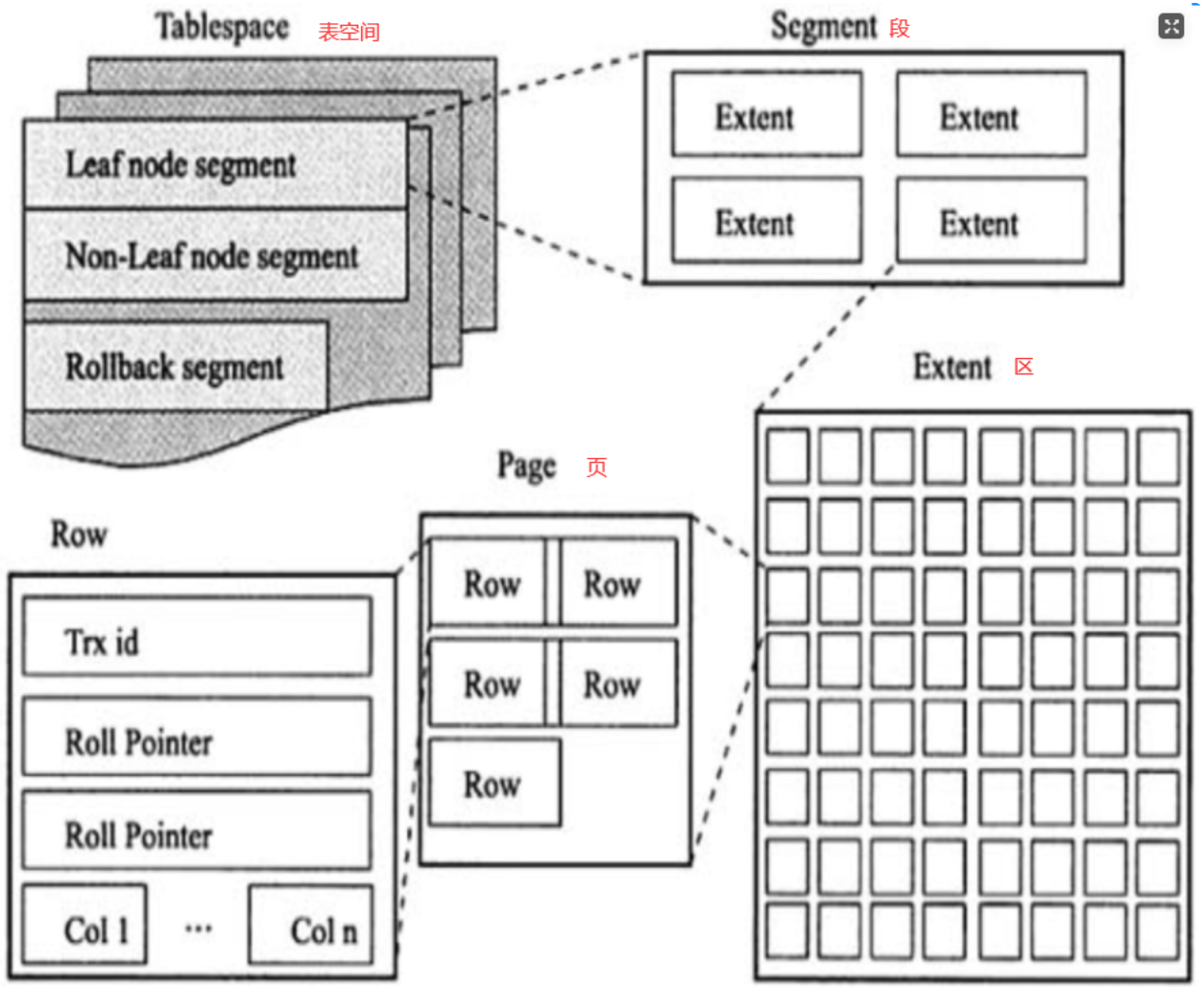
4. InnoDB体系结构---物理存储结构详解

4.0 段

4.1 区

4.2 页


Extents
区（簇，段）
1Mb
64个连续的page，组成一个1mb的区
Mysql逻辑层
我们的视角mysql物理层和linux文件系统是同一层
mysql存储引擎Innodb会将idb文件再进行格式化
t1.ibd
t1.ibd
Page
16kb
Innodb
Mysql物理层
4个连续的4kb的块，组成一个16kb的数据页
1.分区 fdisk /dev/sdb
2.格式化文件类型 mkfs.xfs /dev/sdb
3.挂载使用 mount /dev/sdb /mnt
块
block 4kb
Linux操作系统层
8个连续的512b的扇区，组成一个4kb的块
1kb=1024b(字节)
4*1024/512=8
512b
512b
512b
真实物理层
/dev/sdb



4.3 表

8.0 以前 InnoDB表：

ibd： 数据和索引

frm：存私有的数据字典信息

ibdataN：系统的数据字典信息

8.0 在数据字典改变

> ibd：数据和索引+ 冗余的SDI私有数据字典信息

> 取消了 ibdata中的系统数据字典信息。

> mysql.ibd ---> 整个系统的数据字典，不再放在ibdata1

> sdi序列化的数据字典----> 每个表的表空间自行管理json格式的私有数据字典信息，用来替换frm的。

4.4 索引

4.5 表空间

1.System Tablespace(共享表空间)

> ## 存储方式
>
> ibdata1~ibdataN, 5.5版本默认的表空间类型。
>
> ## ibdata1共享表空间在各个版本的变化

> 5.5版本：
>
> 系统相关：（全局）数据字典信息（表基本结构信息、状态、系统参数、属性..）、UNDO回滚信息
>
> (记录撤销操作)、Double Write buffer信息、临时表信息、change buffer
>
> 用户数据： 表数据行、表的索引数据
>
> 5.6版本：共享表空间只存储于系统数据，把用户数据独立了。
>
> 系统相关：（全局）数据字典信息、UNDO回滚信息、Double Write信息、临时表信息、change
>
> buffer
>
> 5.7版本：在5.6基础上，把临时表独立出来，UNDO也可以设定为独立
>
> 系统相关：（全局）数据字典信息、UNDO回滚信息、Double Write信息、change buffer
>
> 8.0.19版本：在5.7的基础上将UNDO回滚信息默认独立，数据字典不再集中存储了。
>
> 系统相关：Double Write信息、change buffer
>
> 8.0.20版本:在之前版本基础上，独立 Double Write信息
>
> 系统相关：change buffer
>
> https://dev.mysql.com/doc/refman/5.7/en/innodb-architecture.html <https://dev.mysql.com/doc/refman/5.7/en/innodb-architecture.html>
>
> https://dev.mysql.com/doc/refman/8.0/en/innodb-architecture.html <https://dev.mysql.com/doc/refman/8.0/en/innodb-architecture.html>
>
> 总结： 对于InnoDB(8.0)表来讲，例如 city表
>
> city.ibd
>
> mysql.ibd
>
> undo
>
> ibdata
>
> redo
>
> ## 共享表空间管理
>
> ### 扩容共享表空间
>
> mysql> select @@innodb_data_file_path;
>
> +--------------------------+
>
> | @@innodb_data_file_path |
>
> +--------------------------+
>
> | ibdata1:12M:autoextend |

```
+--------------------------+
1 row in set (0.00 sec)
mysql> select @@innodb_autoextend_increment;
+---------------------------------+
| @@innodb_autoextend_increment |
+---------------------------------+
| 64 |
+---------------------------------+
1 row in set (0.00 sec)
```

参数用途：ibdata1文件，默认初始大小12M，不够用会自动扩展，默认每次扩展64M

设置方式：

错误的方法：

innodb_data_file_path=ibdata1:12M;ibdata2:100M;ibdata3:100M:autoextend

重启数据库报错，查看日志文件

vim /data/3306/data/db01.err

```
###################
[ERROR] InnoDB: The innodb_system data file './ibdata1' is of a different
size 4864 pages (rounded down to MB) than the 768 pages specified in the
.cnf file!
###################
```

实际大小：

4864*16K/1024=76M

my.cnf文件设置：

768*16K/1024=12M

正确的方法：

先查看实际大小：

```
[root@db01 data]# ls -lh ibdata1
-rw-r----- 1 mysql mysql 76M May 6 17:11 ibdata1
```

配置文件设定为和实际大小一致：

innodb_data_file_path=ibdata1:76M;ibdata2:100M;ibdata3:100M:autoextend

模拟在初始化时设置共享表空间（生产建议）

5.7 中建议：设置共享表空间2-3个，大小建议512M或者1G，最后一个定制为自动扩展。

8.0 中建议：设置1个就ok，大小建议512M或者1G

清理数据

[root@db01 data]# /etc/init.d/mysqld stop

[root@db01 data]# rm -rf /data/3306/data/*

[root@db01 data]# vim /etc/my.cnf

修改

innodb_data_file_path=ibdata1:100M;ibdata2:100M;ibdata3:100M:autoextend

重新初始化

[root@db01 data]# mysqld --initialize-insecure --user=mysql --

basedir=/data/app/mysql --datadir=/data/3306/data

重启数据库生效

[root@db01 data]# /etc/init.d/mysqld start

2.File-Per-Table Tablespaces（独立表空间）

介绍

5.6版本中，针对用户数据，单独的存储管理。存储表的数据行和索引。

通过参数控制：

mysql> select @@innodb_file_per_table;

+-----------------------+

| @@innodb_file_per_table |

+-----------------------+

| 1 |

+-----------------------+

测试： 共享表空间存储用户数据

mysql> set global innodb_file_per_table=0;

利用独立表空间进行快速数据迁移

源端:3306/test/t100w -----> 目标端：3307/test/t100w

1. 锁定源端t100w表

mysql> flush tables test.t100w with read lock;

```
mysql> show create table test.t100w;
CREATE TABLE `t100w` (
`id` int(11) DEFAULT NULL,
`num` int(11) DEFAULT NULL,
`k1` char(2) DEFAULT NULL,
`k2` char(4) DEFAULT NULL,
`dt` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
2. 目标端创建test库和t100w空表
mysql> create database test charset=utf8mb4;
CREATE TABLE `t100w` (
`id` int(11) DEFAULT NULL,
`num` int(11) DEFAULT NULL,
`k1` char(2) DEFAULT NULL,
`k2` char(4) DEFAULT NULL,
`dt` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
3. 单独删除空的表空间文件
mysql> alter table test.t100w discard tabl      espace;
4. 拷贝源端ibd文件到目标端目录，并设置权限
[root@db01 test]# cp /data/3306/data/test/t100w.ibd /data/3307/data/test/
[root@db01 test]# chown –R mysql.mysql /data/*
5. 导入表空间
mysql> alter table test.t100w import tablespace;
mysql> select count(*) from test.t100w;
+----------+
| count(*) |
+----------+
```

| 1000000 |

6. 解锁源端数据表

mysql> unlock tables;

应用场景：

案例1：同学由于不可抗力因素，导致只剩下test库下的ibd 和 frm文件了。（5.6版本）。没有备份

案例2：同学将ibdata1（5.7版本）误rm掉了。导致只剩下test库下的ibd 和 frm文件了。备份坏 的。

3.Undo tablespace（undo表空间）

1. 作用： 用来作撤销工作。

2. 存储位置： 5.7版本，默认存储在共享表空间中（ibdataN）。8.0版本以后默认就是独立的（undo_001-undo_002）。

3. 生产建议： 5.7版本后，将undo手工进行独立。

4. undo 表空间管理

4.1 如何查看undo的配置参数

SELECT @@innodb_undo_tablespaces; ---->3-5个 #打开独立undo模式，并设置undo的个数。

SELECT @@innodb_max_undo_log_size; #undo日志的大小，默认1G。

SELECT @@innodb_undo_log_truncate; #开启undo自动回收的机制（undo_purge）。

SELECT @@innodb_purge_rseg_truncate_frequency; #触发自动回收的条件，单位是检测次数。

4.2 配置undo表空间

#########官方文档说明#############

Important

The number of undo tablespaces can only be configured

when initializing a MySQL instance and is fixed for the life of the

instance.

###################################

[root@db01 tmp]# /etc/init.d/mysqld stop

[root@db01 tmp]# rm -rf /data/3306/data/*

vim /etc/my.cnf

添加参数

innodb_undo_tablespaces=3

innodb_max_undo_log_size=128M

innodb_undo_log_truncate=ON

innodb_purge_rseg_truncate_frequency=32

重新初始化数据库生效

[root@db01 data]# mysqld --initialize-insecure --user=mysql --

basedir=/data/app/mysql --datadir=/data/3306/data

启动数据库

[root@db01 data]# /etc/init.d/mysqld start

[root@db01 data]# ll /data/3306/data/undo00*

-rw-r----- 1 mysql mysql 10485760 May 7 15:39 /data/3306/data/undo001

-rw-r----- 1 mysql mysql 10485760 May 7 15:39 /data/3306/data/undo002

-rw-r----- 1 mysql mysql 10485760 May 7 15:39 /data/3306/data/undo003

注： 8.0 undo表空间与5.7稍有区别，可参考：

https://dev.mysql.com/doc/refman/8.0/en/innodb-undo-tablespaces.html <https://dev.mysql.com/doc/refman/8.0/en/innodb-undo-tablespaces.html>

1. 添加UNDO

CREATE UNDO TABLESPACE oldguo ADD DATAFILE 'oldguo.ibu';

2. 查看

SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES WHERE

FILE_TYPE LIKE 'UNDO LOG';

3. 删除undo

ALTER UNDO TABLESPACE oldguo SET INACTIVE;

DROP UNDO TABLESPACE oldguo;
SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES WHERE

FILE_TYPE LIKE 'UNDO LOG';

说明： 关于UNDO回收策略

SELECT @@innodb_purge_rseg_truncate_frequency;

4.tmp tablespace（临时表空间）

> 1. 作用： 存储临时表。
>
> 2. 管理：
>
> innodb_temp_data_file_path=ibtmp1:128M;ibtmp2:128M:autoextend:max:500M
>
> 重启生效。
>
> 3. 建议数据初始化之前设定好，一般2-3个，大小512M-1G。

4.6 行格式

> MySQL目前有4种行格式：Redundant、Compact、Dynamic、Compressed
>
> Version 5.6 已经默认使用 Compact
>
> Version 5.7+ 默认使用Dynamic
>
> 结构说明：
>
> 1、存储单元为页（page），16KB（16384B）
>
> 2、每页至少需要存两条数据
>
> 3、每条记录都会有记录头
>
> 4、每条记录3个隐藏列（rowId，transactionId，rollPointer）
>
> 5、记录按照聚簇索引组织存储

> Compact：
>
> 变长字段(记录的长度)列表 + NULL列表 + 记录头信息 + 列值
>
> 变长字段（记录的长度）列表：采用1-2个字节来表示一个字段的长度
>
> 记录头信息：使用5个字节来表示，主要包含：该记录是否被删除，记录类型，下一条记录的相对偏移
>
> 量；
>
> 隐藏列： rowId，transactionId，rollPointer
>
> Dynamic：
>
> 与Compact行格式很像，差异在于页面溢出的处理上；
>
> Compressed：
>
> 在于Dynamic使用了压缩算法；
>
> 页溢出：
>
> 因为每页16KB，至少存储两行，所以每行大概有8KB的数据；抛开记录头信息等，大致每列超过768B就
>
> 会产生页溢出；
>
> Compact：
>
> 1、会将溢出的数据单独放入一个页；外加20B存储额外页的信息（plus the 20-byte pointer to
>
> the externally stored part）
>
> 2、索引可以使用前768B
>
> Dynamic：
> 1、如果页溢出，则使用20B存储整个列信息(列数据都存储在溢出页上)（with the clustered
>
> index record containing only a 20-byte pointer to the overflow page）
>
> 2、可以使用前3072B字符的索引（--innodb-large-prefix决定）
>
> 建议：
>
> 5.7+ 版本Dynamic。
>
> 建议，大字段不要存储到MySQL，最好MongDB相关数据库

4.7 DWB

> 作用：
> MySQL，最小IO单元page（16KB），OS中最小的IO单元是block（4KB）
> 为了防止出现以下问题：
> mysqld process crash in the middle of a page write
> DWB每次1M，2次写完。数据页再刷盘。

4.8 redo日志

> — what?
>
> • DML(8.0+ DDL)操作导致的页面变化，均需要记录Redo日志；
>
> • 大部分为物理日志；
>
> — when?
>
> • 在页面修改完成之后，在脏页刷出磁盘之前，写入Redo日志；
>
> • 日志先行，日志一定比数据页先写回磁盘（WAL）；

• 聚簇索引/二级索引/Undo页面修改，均需要记录Redo日志；

1. 作用: 记录数据页的变化。实现“前进”的功能。WAL（write ahead log），MySQL保证redo优

先于数据写入磁盘。

2. 存储位置： 数据路径下，进行轮序覆盖记录日志

ib_logfile0 48M

ib_logfile1 48M

3. 管理：

3.1 查询redo log文件配置

mysql> show variables like '%innodb_log_file%';

+---------------------------+-----------+

| Variable_name | Value |

+---------------------------+-----------+

| innodb_log_file_size | 50331648 |

| innodb_log_files_in_group | 2 |

+---------------------------+-----------+

3.2 设置

生产建议：

大小： 512M-4G

组数： 2-4组

vim /etc/my.cnf

添加参数：

innodb_log_file_size=100M

innodb_log_files_in_group=3

#重启生效

[root@db01 data]# /etc/init.d/mysqld restart

[root@db01 data]# ll /data/3306/data/ib_logfile*

-rw-r----- 1 mysql mysql 104857600 May 7 16:17 /data/3306/data/ib_logfile0

-rw-r----- 1 mysql mysql 104857600 May 7 16:17 /data/3306/data/ib_logfile1

-rw-r----- 1 mysql mysql 104857600 May 7 16:17 /data/3306/data/ib_logfile2

[root@db01 data]#

4.9 undo日志

> 一 what?
>
> • DML(8.0+ DDL)操作导致的数据记录变化，均需要将记录的前镜像写入Undo日志；
>
> • 逻辑日志；
>
> 一 when?
>
> • DML(8.0 + DDL) 操作修改聚簇索引前，记录Undo日志(Undo日志，先于Redo日志。)；
>
> • 二级索引记录的修改，不记录Undo日志；
>
> • 注意：Undo页面的修改，同样需要记录Redo日志；

5.InnoDB磁盘结构与内存结构工作关系

5.1 LSN（日志序列号）

> 定义说明:
>
> LSN（log sequence number）日志序列号，5.6.3之后占用8字节，LSN主要用于发生crash
>
> 时对数据进行recovery，LSN是一个一直递增的整型数字，表示事务写入到日志的字节总量。
>
> LSN不仅只存在于重做日志中，在每个数据页头部也会有对应的LSN号，该LSN记录当前页最后一次
>
> 修改的LSN号，用于在recovery时对比重做日志LSN号决定是否对该页进行恢复数据。前面说的
>
> checkpoint也是有LSN号记录的，LSN号串联起一个事务开始到恢复的过程。
>
> 查看lsn:
>
> show engine innodb status\G
>
> Log sequence number 2687274848548
>
> Log flushed up to 2687274848516
>
> Pages flushed up to 2687273963960
>
> Last checkpoint at 2687273963960
>
> 简单说明:
>
> Log sequence number: 当前系统最大的LSN号
>
> log flushed up to:当前已经写入redo日志文件的LSN
>
> pages flushed up to：已经将更改写入脏页的lsn号
>
> Last checkpoint at就是系统最后一次刷新buffer pool脏中页数据到磁盘的checkpoint

以上4个LSN是递减的:

LSN1>=LSN2>=LSN3>=LSN4.

内容:

每个数据页有LSN，重做日志有LSN，checkpoint有LSN。

5.2 checkpoint

checkpoint

sharp checkpoint：

完全检查点，数据库正常关闭时，会触发把所有的脏页都写入到磁盘上(这时候logfile的日志就没用

了，脏页已经写到磁盘上了)。

fuzzy checkpoint：

模糊检查点，部分页写入磁盘。
1. master thread checkpoint

差不多以每秒或每十秒的速度从缓冲池的脏页列表中刷新一定比例的页回磁盘，这个过程是异步的，不会

阻塞用户查询。

```
show variables like '%io_cap%';
+------------------------+-------+
| Variable_name | Value |
+------------------------+-------+
| innodb_io_capacity | 200 |
| innodb_io_capacity_max | 2000 |
```

2、flush_lru_list checkpoint

```
mysql> show variables like '%lru%depth';
+----------------------+-------+
| Variable_name | Value |
+----------------------+-------+
| innodb_lru_scan_depth | 1024 |
+----------------------+-------+
1 row in set (0.01 sec)
```

参数innodb_lru_scan_depth控制lru列表中可用页的数量，默认是1024。

3、 async/sync flush checkpoint

log file快满了，会批量的触发数据页回写，这个事件触发的时候又分为异步和同步，不可被覆盖

的redolog占log file的比值：75%--->异步、90%--->同步。

指的是重做日志文件不可用的情况，这时需要强制将一些页刷新回磁盘，而此时脏页是从脏页列表中选取

的。若将已经写入到重做日志的LSN记为redo_lsn，将已经刷新回磁盘最新页的LSN记为

checkpoint_lsn，则可定义：

checkpoint_age = redo_lsn – checkpoint_lsn

再定义以下的变量：

async_water_mark = 75% * total_redo_log_file_size

sync_water_mark = 90% * total_redo_log_file_size

若每个重做日志文件的大小为1GB，并且定义了两个重做日志文件，则重做日志文件的总大小为2GB。那

么async_water_mark=1.5GB，sync_water_mark=1.8GB。则：

当checkpoint_age<async_water_mark时，不需要刷新任何脏页到磁盘；

当async_water_mark<checkpoint_age<sync_water_mark时触发Async Flush，从Flush列表

中刷新足够的脏页回磁盘，使得刷新后满足checkpoint_age<async_water_mark；

checkpoint_age>sync_water_mark这种情况一般很少发生，除非设置的重做日志文件太小，并且

在进行类似LOAD DATA的BULK INSERT操作。此时触发Sync Flush操作，从Flush列表中刷新足够

的脏页回磁盘，使得刷新后满足checkpoint_age<async_water_mark。

4、dirty page too much checkpoint

脏页太多检查点，为了保证buffer pool的空间可用性的一个检查点。

```
mysql> show global status like 'Innodb_buffer_pool_pages%t%';
+----------------------------------+-------+
| Variable_name | Value |
+----------------------------------+-------+
| Innodb_buffer_pool_pages_data | 2964 |
| Innodb_buffer_pool_pages_dirty | 0 |
| Innodb_buffer_pool_pages_total | 8191 |
+----------------------------------+-------+
3 rows in set (0.00 sec)
mysql> show global status like '%wait_free';
```

```
+------------------------------+--------+
| Variable_name | Value |
+------------------------------+--------+
| Innodb_buffer_pool_wait_free | 0 |
+------------------------------+--------+
1 row in set (0.00 sec)
1、Innodb_buffer_pool_pages_dirty/Innodb_buffer_pool_pages_total：表示脏页在
buffer 的占比
2、Innodb_buffer_pool_wait_free：如果>0，说明出现性能负载，buffer pool中没有干净可
用块
2、脏页控制参数
mysql> show variables like '%dirty%pct%';
+--------------------------------+-----------+
| Variable_name | Value |
+--------------------------------+-----------+
| innodb_max_dirty_pages_pct | 75.000000 |
| innodb_max_dirty_pages_pct_lwm | 0.000000 |
+--------------------------------+-----------+
2 rows in set (0.01 sec)
1、默认是脏页占比75%的时候，就会触发刷盘，将脏页写入磁盘，腾出内存空间。建议不调，调太低的
话，io压力就会很大，但是崩溃恢复就很快；
2、lwm：low water mark低水位线，刷盘到该低水位线就不写脏页了，0也就是不限制。
```

5.3 CR(Crash Recovery)

%E4%BD%93%E7%B3%BB%E7%BB%93%E6%9E%84%E2%88%9A%20%7C%201.%E5%AD%98%E5%82%A8%E5%BC%95%E6%93%8E%E7%AE%80%E4%BB%8B1.0%20%E5%AD%98%E5%82%A8%E5%I