ODF IT大咖说 知识分享平台

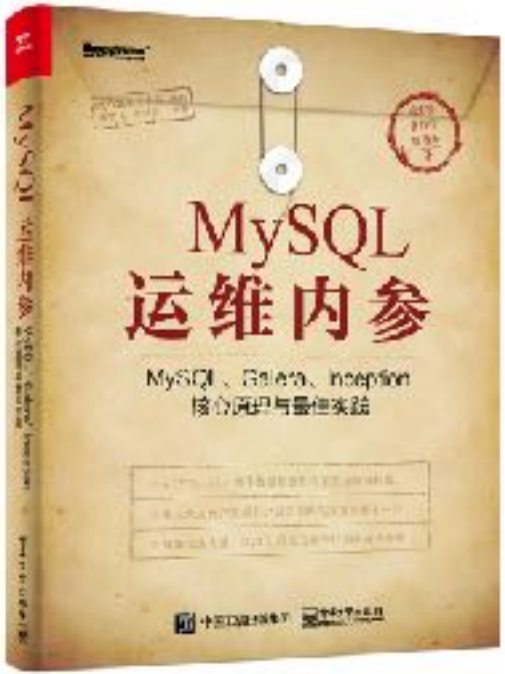
ODF 2017 开源数据库论坛(北京)

开源数据库正在改变世界

2017年9月21日-25日 北京·东仪大酒店

About me

- dameng.com
  - DM database 7 kernel developer
- renren.com
  - MySQL
- qunar.com
  - MySQL DBA
  - inception toolkit
  - redis patch
  - Galera cluster sentinel



向数据安全妥协的产物

——REDO日志详解

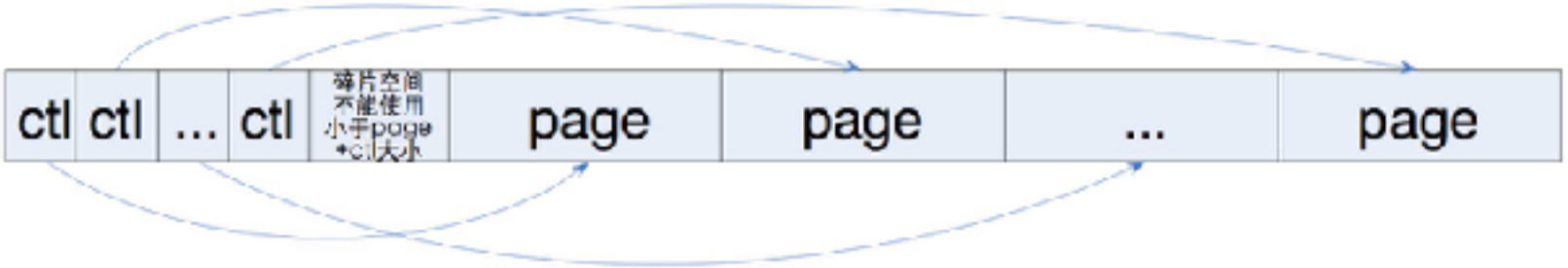
王竹峰 去哪儿网DBA

Topics

- 日志生成
- 日志文件/格式
- Redo及其意义
- Undo

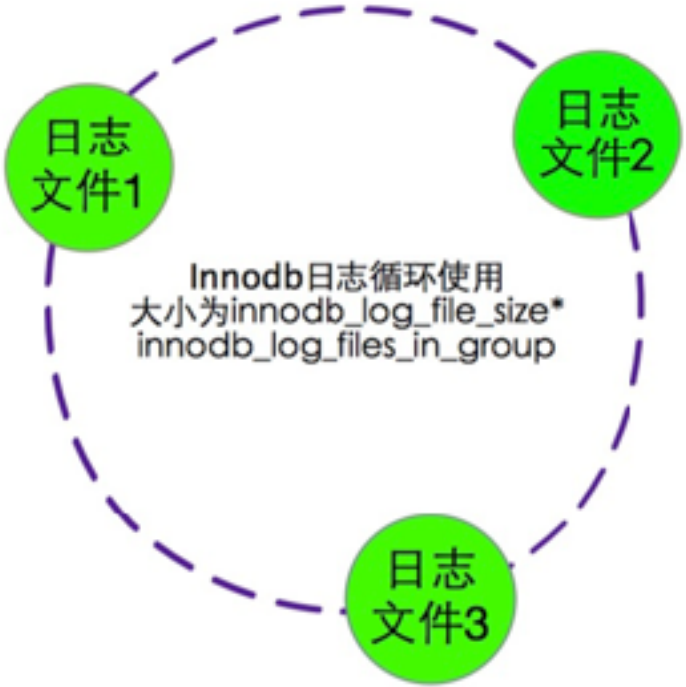
Buffer Pool

- 多实例管理
  - 单个实例空间连续
  - Flush list, LRU list
  - LSN



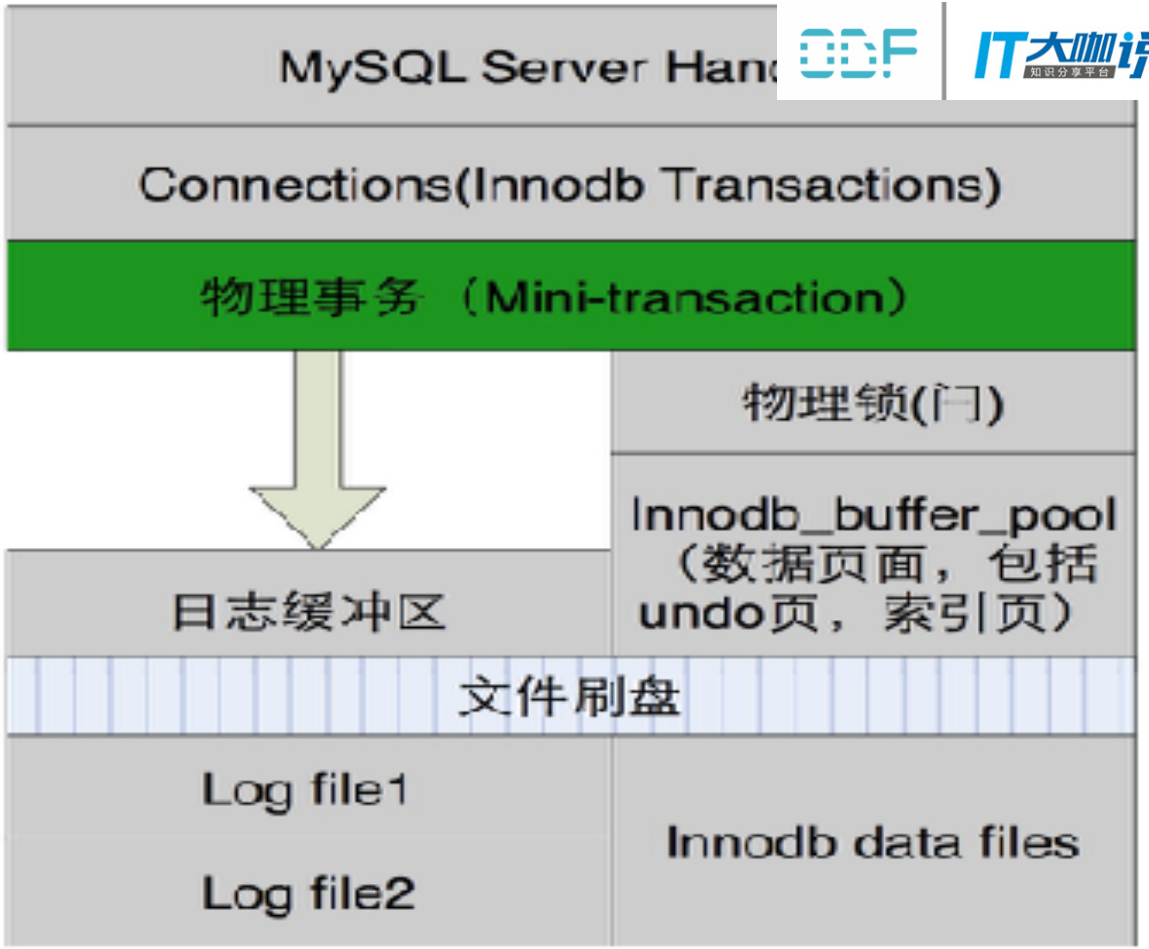
InnoDB LSN

LSN： log sequence number



物理事务

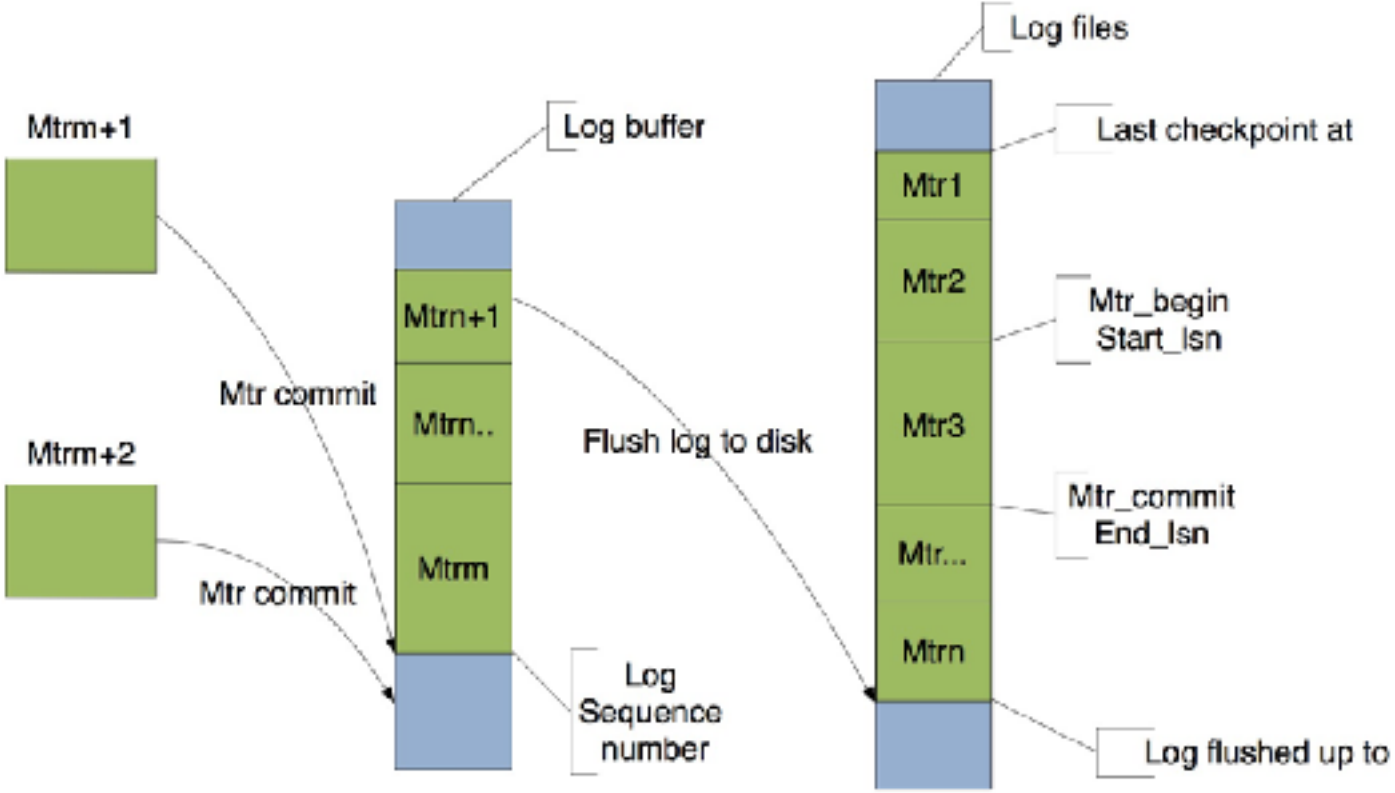
MTR



物理事务MTR

```
struct mtr_struct{
    /* memo stack for locks etc. */
    dyn_array_t memo;

    /* start lsn of the possible logs entry for this mtr */
    ib_uint64_t start_lsn;

    /* end lsn of the possible logs entry for this mtr */
    ib_uint64_t end_lsn;

    /* mini-transaction logs */
    dyn_array_t log;

    /* count of how many page initial logs records have been written to the mtr logs */
    ulint       n_log_recs;

    ulint       log_mode;
};
```

物理事务MTR

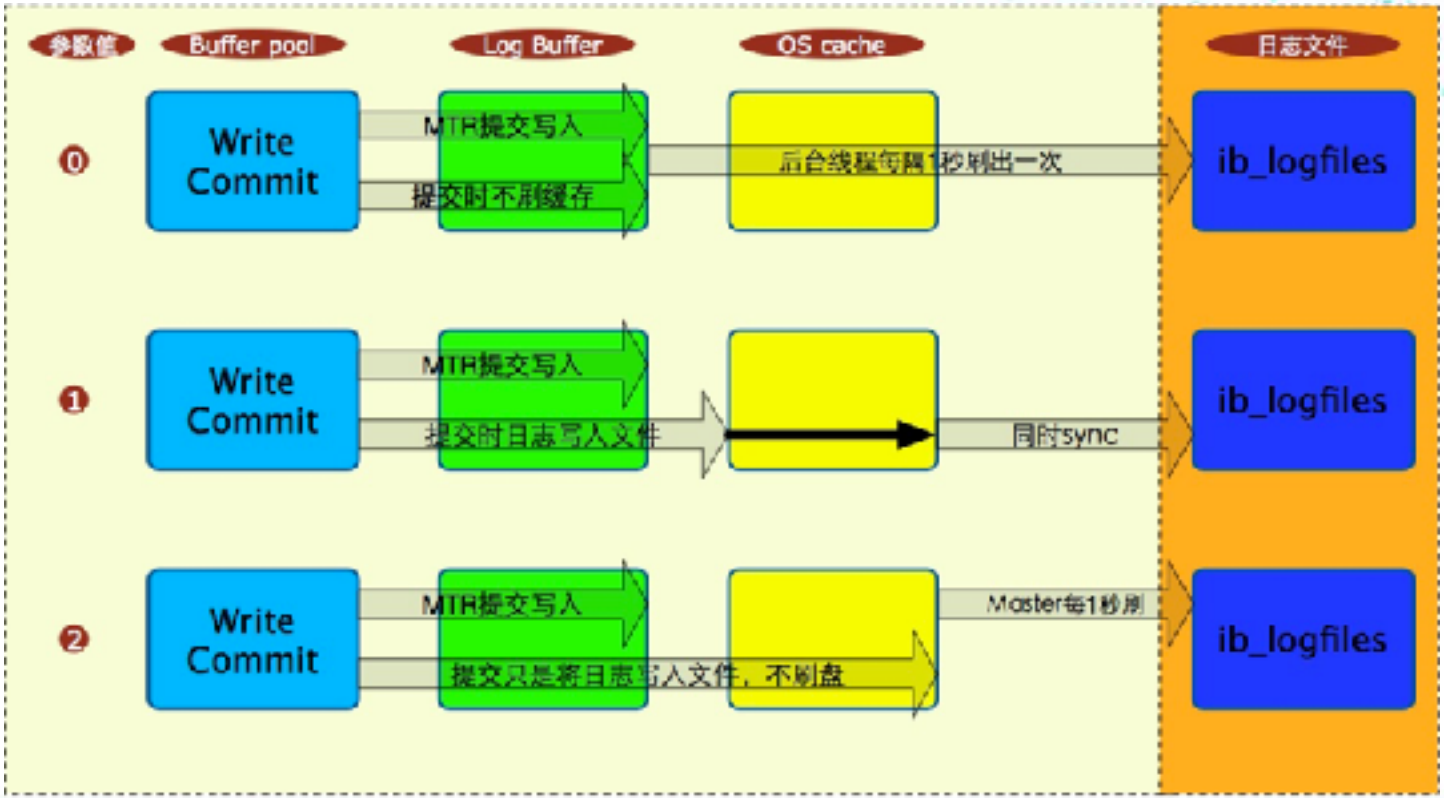

Mtrm+1
Mtrm+2
Mtr commit
Mtr commit
Log buffer
Mtrn+1
Mtrn..
Mtrm
Flush log to disk
Log
Sequence
number
Log files
Mtr1
Mtr2
Mtr3
Mtr...
Mtrn
Last checkpoint at
Mtr_begin
Start_lsn
Mtr_commit
End_lsn
Log flushed up to

物理事务MTR

- innodb_flush_log_at_trx_commit

```
---
LOG
---
Log sequence number 81706062906725
Log flushed up to   81706062867689
Pages flushed up to 81705790802499
Last checkpoint at  81705784758349
```

物理事务MTR


参数值
Buffer pool
Log Buffer
OS cache
日志文件
0
Write Commit
MTR提交写入
提交时不刷缓存
后台线程每隔1秒刷出一次
ib_logfiles
1
Write Commit
MTR提交写入
提交时日志写入文件
同时sync
ib_logfiles
2
Write Commit
MTR提交写入
提交只是将日志写入文件，不刷盘
Master每1秒刷
ib_logfiles

物理事务MTR

- 在mtr日志后面追加MLOG_MULTI_REC_END日志
- 将mtr的日志写入日志缓存区，如果不够则触发日志刷盘。
  - 一个小问题：mtr写的日志理论上应该最多为log_buffer_size的，不然会死循环或者系统崩溃。
- 将脏页加入到buffer的dirty链表中
- 将所有抓住的页面放锁（一个案例）

检查点

- 检查点
  - 定义
  - 过程
  - 与BP关系
  - 数据页面最老LSN

日志文件

- 日志组：逻辑定义，多个文件，只支持一个组。
- 日志文件：物理文件，有自己的格式。
- 容量：现在支持的日志文件空间容量最大是512G（5.6版）
- 页面大小：日志文件页面大小为512B

日志文件

第0页面	0	LOG_GROUP_ID
	4	LOG_FILE_START_LSN——文件LSN
	12	LOG_FILE_NO——archived log file number
	16	LOG_FILE_WAS_CREATED_BY_HOT_BACKUP——归档

日志文件

第1页面	0	LOG_CHECKPOINT_NO——检查点序号
	8	LOG_CHECKPOINT_LSN——检查点lsn
	16	LOG_CHECKPOINT_OFFSET_LOW32——检查点文件偏移低32
	20	LOG_CHECKPOINT_LOG_BUF_SIZE——log buffer size
	24	LOG_CHECKPOINT_ARCHIVED_LSN——归档
	32	LOG_CHECKPOINT_GROUP_ARRAY=LOG_CHECKPOINT_GROUP_ARRAY + LOG_MAX_N_GROUPS * 8=288 存储所有日志组的文件号及偏移，没用
	288	LOG_CHECKPOINT_CHECKSUM_1——校验
	292	LOG_CHECKPOINT_CHECKSUM_2——校验
	296	LOG_CHECKPOINT_FSP_FREE_LIMIT——废弃
	300	LOG_CHECKPOINT_FSP_MAGIC_N——废弃
	304	LOG_CHECKPOINT_OFFSET_HIGH32——检查点文件偏移高32

日志文件

第2页面	空着
第3页面	同“第1页面”

日志文件

0	LOG_BLOCK_HDR_NO——LSN转换来的一个NO
4	LOG_BLOCK_HDR_DATA_LEN——当前块的数据长度
6	LOG_BLOCK_FIRST_REC_GROUP——块中mtr的开始位置
8	LOG_BLOCK_CHECKPOINT_NO——检查点序列号

普通页面

普通页面

日志记录格式

- 日志记录分为日志记录头和日志记录体

Type	spaceid	offset	data

- 日志头分为，接口mlog_write_initial_log_record_fast：
  - Type：日志记录类型，一字节。最高位
  - Space：表空间，compressed整形，1-5字节
  - Offset：页号，compressed整形，1-5字节
  - Data: 根据不同的TYPE有不同的格式

日志文件

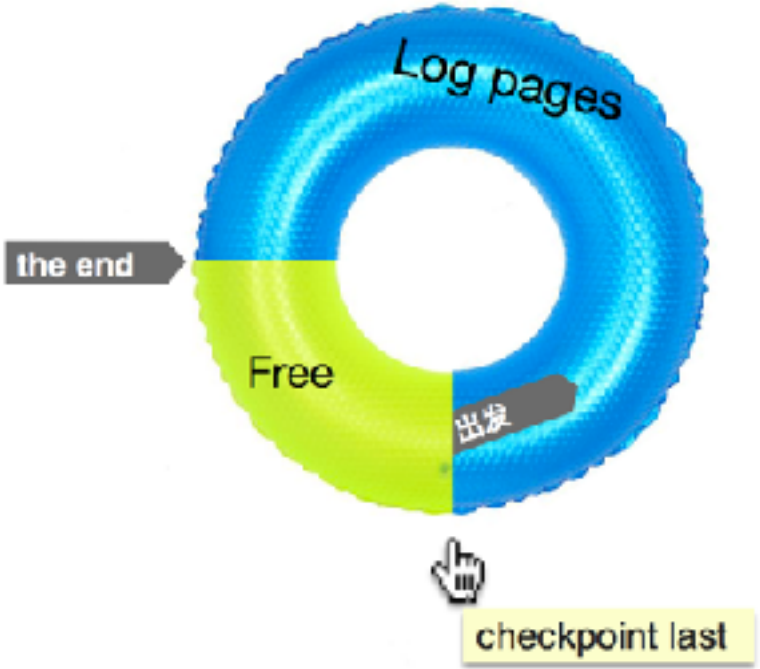
- 日志记录类型
  - 固定长度：MLOG_1BYTE，MLOG_2BYTES，MLOG_4BYTES，MLOG_8BYTES
  - 可变长度：MLOG_WRITE_STRING
  - 逻辑类型：MLOG_COMP_REC_CLUST_DELETE_MAR……
  - 特殊类型：MLOG_MULTI_REC_END（double write的问题）
  - 文件系统：MLOG_INIT_FILE_PAGE、 MLOG_COMP_PAGE_CREATE

日志文件

- 刷盘时机
  - Log_buffer满了。
  - Master 线程后台
  - 执行修改语句时检查，如果需要则做日志刷盘
  - 刷一个页面时，要保证日志已经刷到这个页面的最新LSN处
  - 事务提交时（参数flush_log_at_trx_commit）

日志恢复

- 恢复时写不写日志?
- 扫描两遍?



日志恢复

- 累赘？此话怎讲？
- 可不可以不写日志？
- 提高性能？
- 日志文件设置多大合适？

日志文件

- 日志的意义。
- 为什么要恢复?
- 相关参数
- 恢复的顺序
- 正常关闭需要恢复么?

日志文件

日志文件

- 扫描HASH表
- MTR处理一个页面
- 累赘的价值(不留痕迹)

事务回滚

- 为什么需要回滚?
- 事务ID分配
- 回滚段分类
- 最大事务个数
- 历史信息

事务回滚


回滚段组织架构
回滚段1
回滚段2
回滚段
回滚段128
undo 段1
undo 段...
undo 段1204
undo 页面链表

事务回滚

undo page

TRX_UNDO_PAGE_TYPE
TRX_UNDO_PAGE_START
TRX_UNDO_PAGE_FREE
TRX_UNDO_PAGE_NODE

undo segment

TRX_UNDO_STATE
TRX_UNDO_LAST_LOG
TRX_UNDO_FSEG_HEADER
TRX_UNDO_PAGE_LIST

undo log

TRX_UNDO_TRX_ID
TRX_UNDO_TRX_NO
TRX_UNDO_DEL_MARKS
TRX_UNDO_LOG_START
TRX_UNDO_XID_EXISTS
TRX_UNDO_DICT_TRANS
TRX_UNDO_TABLE_ID
TRX_UNDO_NEXT_LOG
TRX_UNDO_PREV_LOG
TRX_UNDO_HISTORY_NODE

事务回滚

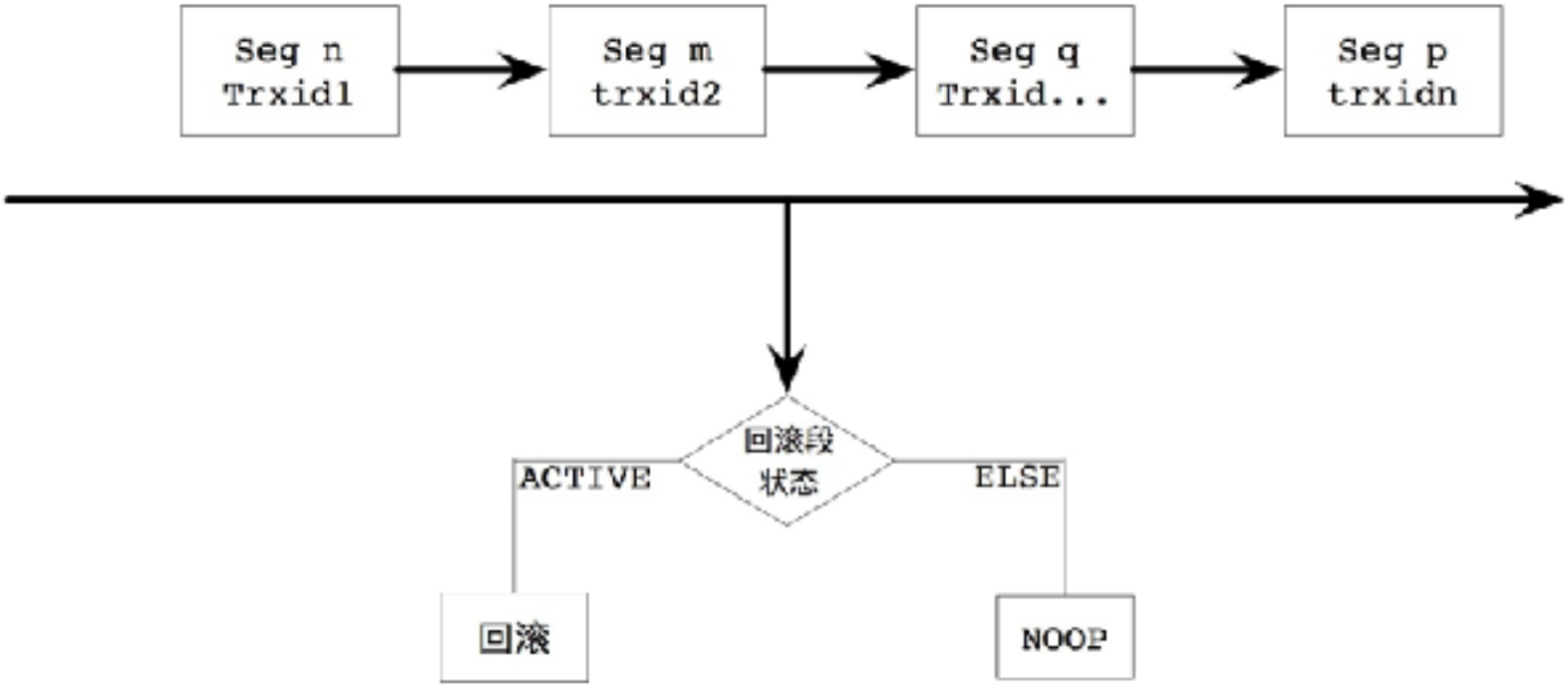
- 像REDO一样，UNDO也有不同的类型：
  - TRX_UNDO_INSERT_REC
  - TRX_UNDO_UPD_EXIST_REC
  - TRX_UNDO_UPD_DEL_REC
  - TRX_UNDO_DEL_MARK_REC
  - TRX_UNDO_CMPL_INFO_MULT
  - TRX_UNDO_UPD_EXTERN

事务回滚

- 举例：TRX_UNDO_INSERT_REC
  - End of rec：记录结尾的页内偏移
  - Insert Type：TRX_UNDO_INSERT_REC
  - Undo No：
  - Table ID：表ID
  - Start of rec：记录开头的页内偏移
  - 记录不会跨页面

end of rec 2byte	undo type 1byte	undo no Compressed int64	table id Compressed int64	trx_id Compressed int64	roll_ptr Compressed int64	n_unique*(fld_len+fld_data) n_unique*(Compressed int+datalen)	start of rec 2byte

回滚时刻


Seg n
Trxid1
Seg m
trxid2
Seg q
Trxid...
Seg p
trxidn
回滚段
状态
ACTIVE
ELSE
回滚
NOOP

Undo与Redo的关系

- Undo页面与B树页面的关系
- 写入数据走buffer
- 写REDO日志
- 专门的日志记录类型
- REDO数据恢复
- 先做REDO，后做UNDO。

Thanks

关注开源数据库论坛