第二十七讲：InnoDB 的 UndoLog 数据结构设计

知春路遇上八里桥

<2024-10-09 Wed>

1. UndoLog 引入
2. 系统表空间
3. 回滚段
4. UNDO_LOG 页
5. UNDO 记录

1

UndoLog 引入

事务特性 ACID 特性 [1]

- 原子性 (Atomicity)：事务中的所有操作要么全部完成，要么全部不执行
  - `autocommit` 自动提交的设置
  - `COMMIT` 提交语句，`ROLLBACK` 回滚语句
- 一致性 (Consistency)：事务在执行前后，数据库的完整性约束不会被破坏
  - Double Write 双写机制
  - Crash Recovery 崩溃恢复机制
- 隔离性 (Isolation)：多个事务并发执行时，一个事务的执行不会受到其他事务的干扰
  - `autocommit` 自动提交的设置
  - `SET TRANSACTION` 开启事务，事务的隔离级别设置
    1. SERIALIZABLE
    2. READ UNCOMMITTED
    3. READ COMMITTED (RC)
    4. REPEATABLE READ (RR)
  - 数据库底层锁，lock `data_locks` 和 `data_lock_waits`
- 持久性 (Durability)：提交后的事务对数据库的修改是永久性的，即使系统发生故障也不会丢失
  - `innodb_flush_log_at_trx_commit`
  - `sync_binlog`
  - `fsync()` 系统调用
  - …

[1] https://dev.mysql.com/doc/refman/8.0/en/mysql-acid.html

UndoLog 的意义

- WAL (Write-Ahead Logging)
  - 在修改记录之前，把该记录的原值（Before Image）先保存起来（Undo Log）再修改
  - 修改过程中出错可以回滚
- MVCC (Multi-Version Concurrency Control) 用来实现高并发读写数据库操作[2]
  - 事务的隔离性，简单的说就是不同活跃事务的数据互相可能是不可见的
  - 每个事务通过 DB_TRX_ID 唯一标记，每个事务都会有自己的快照
  - 每个开启的事务会形成一个 ReadView （读取视图），ReadView 隔离不同事务操作
  - 在事务开始时，InnoDB 会根据当前系统中活跃的事务来创建 ReadView
  - 当多个 ReadView 修改的数据可能会产生冲突，此时就需要锁 (lock) 对访问进行控制
- 行记录中包含以下隐藏字段，这些隐藏字段在 MVCC 控制中有比较大的作用
  - DB_ROW_ID 行记录 ID，聚簇索引的 key，通常会自增
  - DB_TRX_ID 事务 ID，记录最后一个对该行数据操作的事务 ID
  - DB_ROLL_PTR 回滚指针，指向回滚段中的记录
- 崩溃恢复 (crash recovery)
  - 数据库在任何时候都可能发生宕机；包括停电，软硬件 bug
  - UndoLog 在数据库重启时把正在提交的事务完成提交，活跃的事务回滚
  - 保证了事务的原子性，以此来让数据库恢复到一个一致性的状态

[2] https://dev.mysql.com/doc/refman/8.0/en/innodb-multi-versioning.html

2

系统表空间

系统表空间

1. ibdata1 (全局系统表空间) 布局，其中 #5 页包含事务元信息头

```
| offset           | page_no | page_type    | note           |
|------------------+---------+--------------+----------------|
|           0x0(0) |      #0 | FSP_HDR      |                |
|     0x4000(16384) |      #1 | IBUF_BITMAP |                |
|     0x8000(32768) |      #2 | INODE        |                |
|     0xc000(49152) |      #3 | SYS          |                |
|    0x10000(65536) |      #4 | INDEX        |                |
|    0x14000(81920) |      #5 | TRX_SYS      | 事务系统信息头 |
|    0x18000(98304) |      #6 | SYS          | 回滚段头       |
|  0x1c000(114688) |      #7 | SYS          | 数据字典信息   |
|  0x20000(131072) |      #8 | SDI          |                |
|  0x24000(147456) |      #9 | UNKNOWN      |                |
```

2. ibtmp1 (全局临时表空间) 布局，#3 页后续存放回滚段头 [3]

```
| offset           | page_no | page_type    | note           |
|------------------+---------+--------------+----------------|
|           0x0(0) |      #0 | FSP_HDR      |                |
|     0x4000(16384) |      #1 | ALLOCATED    |                |
|     0x8000(32768) |      #2 | INODE        |                |
|     0xc000(49152) |      #3 | SYS          | 回滚段头       |
|    0x10000(65536) |      #4 | SYS          | 回滚段头       |
|    0x14000(81920) |      #5 | SYS          | 回滚段头       |
```

[3] 回滚段 (Rollback Segment) 简称 RSEG

ibdata1 系统表空间定义

- 源码见 ☞ storage/innobase/include/fsp0types.h

```
/** extent descriptor */
constexpr uint32_t FSP_XDES_OFFSET = 0;
/** insert buffer bitmap; The ibuf bitmap pages are the ones whose page number
is the number above plus a multiple of XDES_DESCRIBED_PER_PAGE */
constexpr uint32_t FSP_IBUF_BITMAP_OFFSET = 1;
/** in every tablespace */
constexpr uint32_t FSP_FIRST_INODE_PAGE_NO = 2;

/** The following pages exist in the system tablespace (space 0). */
/** insert buffer header page, in tablespace 0 */
constexpr uint32_t FSP_IBUF_HEADER_PAGE_NO = 3;
/** insert buffer B-tree root page in tablespace 0;
The ibuf tree root page number in tablespace 0; its fseg inode is on the page
number FSP_FIRST_INODE_PAGE_NO */
constexpr uint32_t FSP_IBUF_TREE_ROOT_PAGE_NO = 4;
/** transaction system header, in tablespace 0 */
constexpr uint32_t FSP_TRX_SYS_PAGE_NO = 5;
/** first rollback segment page, in tablespace 0 */
constexpr uint32_t FSP_FIRST_RSEG_PAGE_NO = 6;
/** data dictionary header page, in tablespace 0 */
constexpr uint32_t FSP_DICT_HDR_PAGE_NO = 7;
```

UNDO 表空间

- MySQL 8.0.x 的 UndoLog 独立的默认 Undo 表空间 [4] ，分别为 undo_001 和 undo_002

```
mysql/data » ibr undo_001 list -l 15
page_no=0, page_type=FSP_HDR, space_id=UndoSpace(1), lsn=3902692189, offset=0x0(0)
page_no=1, page_type=IBUF_BITMAP, space_id=UndoSpace(1), lsn=8213, offset=0x4000(16384)
page_no=2, page_type=INODE, space_id=UndoSpace(1), lsn=436780744, offset=0x8000(32768)
page_no=3, page_type=RSEG_ARRAY, space_id=UndoSpace(1), lsn=1239571, offset=0xc000(49152)
page_no=4, page_type=SYS, space_id=UndoSpace(1), lsn=3902713249, offset=0x10000(65536)
page_no=5, page_type=SYS, space_id=UndoSpace(1), lsn=3902713287, offset=0x14000(81920)
page_no=6, page_type=SYS, space_id=UndoSpace(1), lsn=3902713325, offset=0x18000(98304)
page_no=7, page_type=SYS, space_id=UndoSpace(1), lsn=3068606270, offset=0x1c000(114688)
page_no=8, page_type=SYS, space_id=UndoSpace(1), lsn=3068606616, offset=0x20000(131072)
page_no=9, page_type=SYS, space_id=UndoSpace(1), lsn=2100497087, offset=0x24000(147456)
page_no=10, page_type=SYS, space_id=UndoSpace(1), lsn=3902094138, offset=0x28000(163840)
page_no=11, page_type=SYS, space_id=UndoSpace(1), lsn=3902409545, offset=0x2c000(180224)
page_no=12, page_type=SYS, space_id=UndoSpace(1), lsn=3902713379, offset=0x30000(196608)
page_no=13, page_type=SYS, space_id=UndoSpace(1), lsn=3902409621, offset=0x34000(212992)
page_no=14, page_type=SYS, space_id=UndoSpace(1), lsn=3902715338, offset=0x38000(229376)
```

[4] https://dev.mysql.com/doc/refman/8.0/en/innodb-undo-tablespaces.html

3

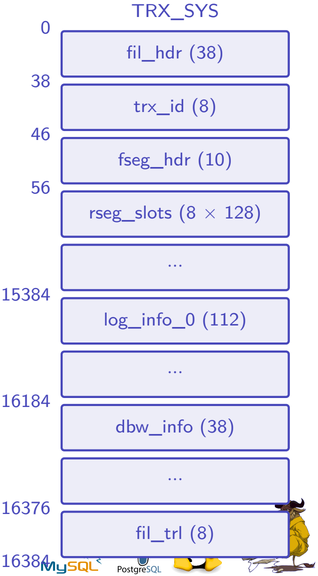
回滚段

TRX_SYS 页

- 源码见 ☞ storage/innobase/include/trx0sys.h

```
/** Transaction system header */
/*------------------------------------------------------------- @{ */
/** the maximum trx id or trx number modulo TRX_SYS_TRX_ID_UPDATE_MARGIN written
   to a file page by any  transaction; the assignment of transaction ids
   continues from  this number rounded up by TRX_SYS_TRX_ID_UPDATE_MARGIN  plus
   TRX_SYS_TRX_ID_UPDATE_MARGIN when the database is started */
constexpr uint32_t TRX_SYS_TRX_ID_STORE = 0;
/** segment header for the  tablespace segment the trx system is created into */
constexpr uint32_t TRX_SYS_FSEG_HEADER = 8;
/** the start of the array of rollback segment specification slots */
constexpr uint32_t TRX_SYS_RSEGS = 8 + FSEG_HEADER_SIZE;
```

- trx_id 事务 ID
- fseg_hdr 所属回滚段
- rseg_slots 指向回滚段页码的数组，每项包含一下两个字段
  - space_id 表空间 ID
  - page_no 页码
- log_info_0 记录 binlog 文件名称
- dbw_info 记录 DoubleWrite 缓冲区信息



TRX_SYS

0 fil_hdr (38)

38 trx_id (8)

46 fseg_hdr (10)

56 rseg_slots (8 × 128)

···

15384 log_info_0 (112)

···

16184 dbw_info (38)

···

16376 fil_trl (8)

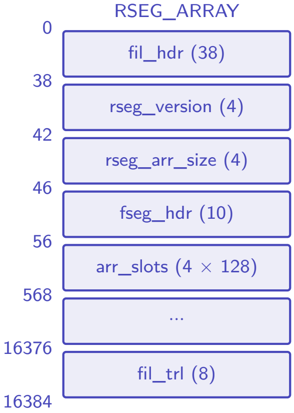
16384

RSEG_ARRAY 页

- 源码见 ☞ storage/innobase/include/trx0rseg.h

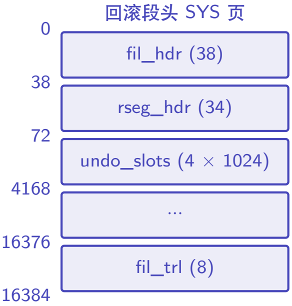
```
/** The RSEG ARRAY base version is a number derived from the string
'RSEG' [0x 52 53 45 47] for extra validation. Each new version
increments the base version by 1. */
constexpr uint32_t RSEG_ARRAY_VERSION = 0x52534547 + 1;

/** The RSEG ARRAY version offset in the header. */
constexpr uint32_t RSEG_ARRAY_VERSION_OFFSET = 0;

/** The current number of rollback segments being tracked in this array */
constexpr uint32_t RSEG_ARRAY_SIZE_OFFSET = 4;
```

- rseg_version 是回滚段的版本
  - 起始版本号为 0x52534547
- rseg_arr_size 当前数组中追溯的回滚段数量
- fseg_hdr 所属回滚段
- arr_slots 指向回滚段头 (RSEG_HDR) 的数组
  - 每个 slot 占 4 个字节，表示一个页号
  - 最多容纳 128 个 slot



回滚段头 SYS 页

- 源码见 ☞ storage/innobase/include/trx0rseg.h
- rseg_hdr 是 Rollback Segment Header
  - 记录回滚段的头信息
- undo_slots 存放活跃事务的 UndoHeader 的页码
  - 每个 slot 占 4 个字节
  - 总共有 1024 个 slot
  - 每个事务会申请一个或两个 slot
  - 通常 INSERT/UPDATE 分开
- 同时把事务的第一个 Undo 页放入对应 slot 中
- InnoDB 理论上的最大事务并发数为
  - $128 \times 128 \times 1024$
  - Undo Tablespace 数量 (128)
  - Rollback Segment 数量 (128)
  - TRX_RSEG_N_SLOTS (1024)

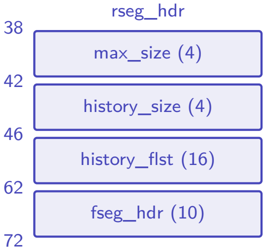


SYS 页的 rseg_hdr

- 源码见 ☞ storage/innobase/include/trx0rseg.h

```
/** Maximum allowed size for rollback segment in pages */
constexpr uint32_t TRX_RSEG_MAX_SIZE = 0;
/** Number of file pages occupied by the logs in the history list */
constexpr uint32_t TRX_RSEG_HISTORY_SIZE = 4;
/* The update undo logs for committed transactions */
constexpr uint32_t TRX_RSEG_HISTORY = 8;
/* Header for the file segment where this page is placed */
constexpr uint32_t TRX_RSEG_FSEG_HEADER = 8 + FLST_BASE_NODE_SIZE;
```

- max_size 回滚段可以有用的最大页数
- history_size 历史列表中包含的页数
- history_flst 历史列表的起始节点
- fseg_hdr File Segement 存放的位置

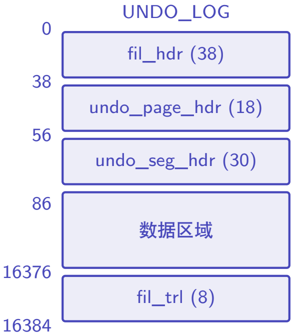


4

UNDO_LOG 页

UNDO_LOG 页

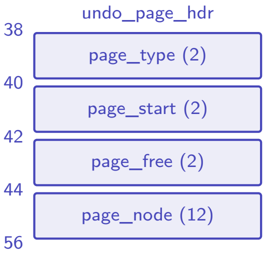
- Undo Page 一般分两种情况
  - HeaderPage：包含 undo_log_hdr，记录元信息
    1. 一个 HeaderPage 同一时刻只隶属于同一个活跃事务
    2. 一个 HeaderPage 内容可能包含多个已经提交的事务和一个活跃事务
  - NormalPage：存储 UndoRecord 数据
    1. 当活跃事务产生的 UndoRecord 超过 HeaderPage 容量后，单独再为此事务分配的 NormalPage
    2. NormalPage 只隶属于一个事务，只包含 undo_page_hdr 不包含 undo_seg_hdr
- undo_page_hdr (Undo Page Header)
  - 记录当前页面的元信息
- undo_seg_hdr (Undo Segment Header)
  - 记录当前页面所属回滚段的元信息
- 数据区域，包含紧密排布的 UndoLog
  - undo_log_hdr (Undo Log Header) 记录日志的元信息
  - undo_rec Undo 记录节点



UNDO_LOG 页的 undo_page_hdr

- page_type 该页事务的类型
  - 包含 TRX_UNDO_INSERT，TRX_UNDO_UPDATE 两种
- page_start 最新事务开始记录 UndoLog 起始位置
- page_free 页内空闲空间起始地址
  - 新申请的 UNDO 日志总是在 free 后面
- page_node 同一事务，UNDO 页的双向链表
- 源码见 ☞ storage/innobase/include/trx0undo.h

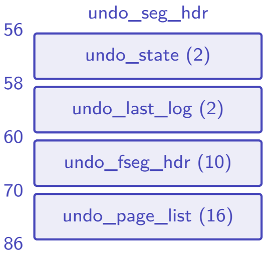
```
/** TRX_UNDO_INSERT or TRX_UNDO_UPDATE */
constexpr uint32_t TRX_UNDO_PAGE_TYPE = 0;
/** Byte offset where the undo log records for the LATEST transaction start on
 this page (remember that in an update undo log, the first page can contain
 several undo logs) */
constexpr uint32_t TRX_UNDO_PAGE_START = 2;
/** On each page of the undo log this field contains the byte offset of the
 first free byte on the page */
constexpr uint32_t TRX_UNDO_PAGE_FREE = 4;
/** The file list node in the chain of undo log pages */
constexpr uint32_t TRX_UNDO_PAGE_NODE = 6;
```



UNDO_LOG 页的 undo_seg_hdr

- undo_state 回滚段的状态，
  - 例如 TRX_UNDO_ACTIVE，TRX_UNDO_CACHED 等
- undo_last_log 当前页最后 UndoLogHeader 的位置
- undo_fseg_hdr 当前段描述符，指向 INode 页
- undo_page_list 回滚段的页面列表
- 源码见 ☞ storage/innobase/include/trx0undo.h

```
/** TRX_UNDO_ACTIVE, ... */
constexpr uint32_t TRX_UNDO_STATE = 0;
/** Offset of the last undo log header on the segment header page, 0 if none */
constexpr uint32_t TRX_UNDO_LAST_LOG = 2;
/** Header for the file segment which the undo log segment occupies */
constexpr uint32_t TRX_UNDO_FSEG_HEADER = 4;
/** Base node for the list of pages in the undo log segment; defined only on the
 undo log segment's first page */
constexpr uint32_t TRX_UNDO_PAGE_LIST = 4 + FSEG_HEADER_SIZE;
```



undo_seg_hdr

56

undo_state (2)

58

undo_last_log (2)

60

undo_fseg_hdr (10)

70

undo_page_list (16)

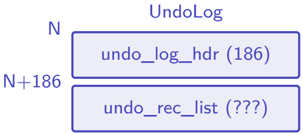
86

5

UNDO 记录

UndoLog

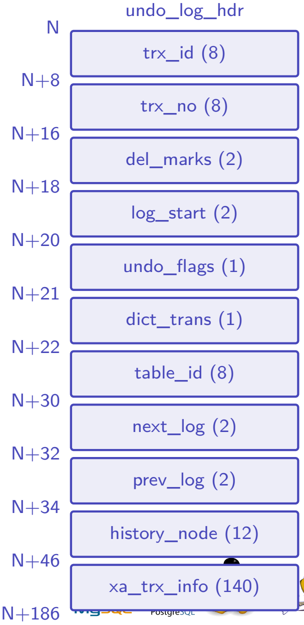
- undo_log_hdr UndoLog 信息头
  - 记录 UndoLog 的基础信息
  - 可能包含 XID, GTID 相关信息
  - undo_seg_hdr.undo_last_log 指定最后的 undo_log_hdr
- undo_rec_list UndoRecord 列表
  - 通常一个 undo_log_hdr 后跟着连续的一个或多个 undo_rec
  - UndoRecord 包含两种： Insert/Update 类型
- undo_log_hdr 和 undo_rec 是紧密排布的
  - 它们的长度都是不定长的



undo_log_hdr

- trx_id 事务 ID 和 trx_no 事务编号
- del_marks 删除标记，如果涉及删除数据则为 1
- log_start 第一个 UndoLog 记录地址
- undo_flags 用来标记 XID, GTID 等信息
- next_log 和 prev_log 本页面其他的 UndoLogHeader 指针
- history_node 事务结束时放入 history 列表的节点
- xa_trx_info 当 undo_flags 标记 XID 所需要的额外信息
- 源码见 ☞ storage/innobase/include/trx0undo.h

```
/** Transaction id */
constexpr uint32_t TRX_UNDO_TRX_ID = 0;
/** Transaction number of the transaction; defined only if the log is in a
 history list */
constexpr uint32_t TRX_UNDO_TRX_NO = 8;
/** Defined only in an update undo log: true if the transaction may have done
 delete markings of records, and thus purge is necessary */
constexpr uint32_t TRX_UNDO_DEL_MARKS = 16;
/** Offset of the first undo log record  of this log on the header page; purge
 may remove undo log record from the log start, and therefore this is not
 necessarily the same as this log header end offset */
constexpr uint32_t TRX_UNDO_LOG_START = 18;
```



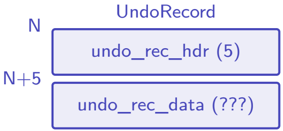
undo_log_hdr

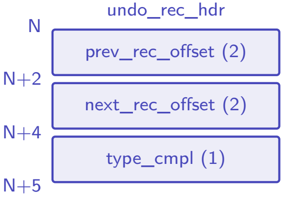
N trx_id (8)
N+8 trx_no (8)
N+16 del_marks (2)
N+18 log_start (2)
N+20 undo_flags (1)
N+21 dict_trans (1)
N+22 table_id (8)
N+30 next_log (2)
N+32 prev_log (2)
N+34 history_node (12)
N+46 xa_trx_info (140)
N+186

UndoRecord

- undo_rec_hdr UndoRecord 信息头
  - prev_rec_offset 上一个 UndoRecord 页面偏移值
  - next_rec_offset 下一个 UndoRecord 页面偏移值
  - type_cmpl 元信息，包含一些标志位
    1. type：INSERT_REC / UPD_EXIST_REC / UPD_DEL_REC / DEL_MARK_REC
    2. compilation info：NO_ORD_CHANGE / NO_SIZE_CHANGE
    3. extern flag 更新外部 BLOB
- undo_rec_data 是 UndoRecord 的信息体
  - 具体的数据载体，分为插入型和更新型
  - UndoRecForInsert 插入类型的 UndoRecord
  - UndoRecForUpdate 更新类型的 UndoRecord
- 代码 ☞ ⋯/trx/trx0rec.cc

```
/** fresh insert into clustered index */
constexpr uint32_t TRX_UNDO_INSERT_REC = 11;
/** update of a non-delete-marked  record */
constexpr uint32_t TRX_UNDO_UPD_EXIST_REC = 12;
/** update of a delete marked record to a not delete marked record; also the
fields of the record can change */
constexpr uint32_t TRX_UNDO_UPD_DEL_REC = 13;
```



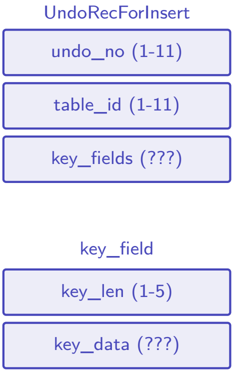


UndoRecForInsert

- undo_no Undo Number 表示事务的第几条 UndoRecord
- table_id Table ID
- key_fields 键值列表，聚簇索引中 key 的列表
  - 数量 n_uniq_key 取决于聚簇索引中 key 的数量
  - key_len 数据的字节数
  - key_data 数据存储
- trx_undo_page_report_insert(..) 函数

☞ ⋯/trx/trx0rec.cc

```
/* Reserve 2 bytes for the pointer to the next undo log record */
ptr += 2;

/* Store first some general parameters to the undo log */
*ptr++ = TRX_UNDO_INSERT_REC;
ptr += mach_u64_write_much_compressed(ptr, trx->undo_no);
ptr += mach_u64_write_much_compressed(ptr, index->table->id);
/*----------------------------------------*/
/* Store then the fields required to uniquely determine the record
to be inserted in the clustered index */

for (i = 0; i < dict_index_get_n_unique(index); i++) {
  const dfield_t *field = dtuple_get_nth_field(clust_entry, i);
  ulint flen = dfield_get_len(field);
```



UndoRecForInsert

undo_no (1-11)

table_id (1-11)

key_fields (???)

key_field

key_len (1-5)

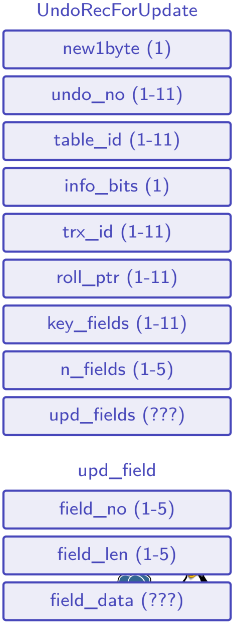
key_data (???)

UndoRecForUpdate

- trx_id 事务 ID 和 roll_ptr 回滚指针
- n_field 更新字段的数量
- upd_fields 更新字段值
    - field_no 字段的编号 Field Number
    - field_len 字段的字节数
    - field_data 字段存储
- trx_undo_page_report_modify(..) 函数

☞ ⋯/trx/trx0rec.cc

```
/* Reserve 2 bytes for the pointer to the next undo log record */
ptr += 2;

/* Store first some general parameters to the undo log */

if (!update) {
  ut_ad(!rec_get_deleted_flag(rec, dict_table_is_comp(table)));
  type_cmpl = TRX_UNDO_DEL_MARK_REC;
} else if (rec_get_deleted_flag(rec, dict_table_is_comp(table))) {
  type_cmpl = TRX_UNDO_UPD_DEL_REC;
```



UndoRecForUpdate

new1byte (1)

undo_no (1-11)

table_id (1-11)

info_bits (1)

trx_id (1-11)

roll_ptr (1-11)

key_fields (1-11)

n_fields (1-5)

upd_fields (???)

upd_field

field_no (1-5)

field_len (1-5)

field_data (???)

结束