

第二十一讲：InnoDB 存储引擎的行记录格式

知春路遇上八里桥

<2024-08-06 Tue>



1 前情提要

2 COMPACT 行记录

3 行记录实验

4 SDI

5 行数据解析



1

前情提要



上节内容

- 逻辑结构

- ▶ 表空间 (Tablespace)
- ▶ 段 (Segment)
- ▶ 区 (Extent)
- ▶ 页 (Page)

- 物理结构

- ▶ 数据文件 (Datafile)
 - ① .ibd InnoDB 数据文件
 - ② .csv 文本格式表格文件
 - ③ .frm 早期 MySQL 记录表结构
- ▶ 页 (Page), `fil_hdr.page_type`
 - ① FSP_HDR 页
 - ② INODE 页
 - ③ INDEX 页, 数据页

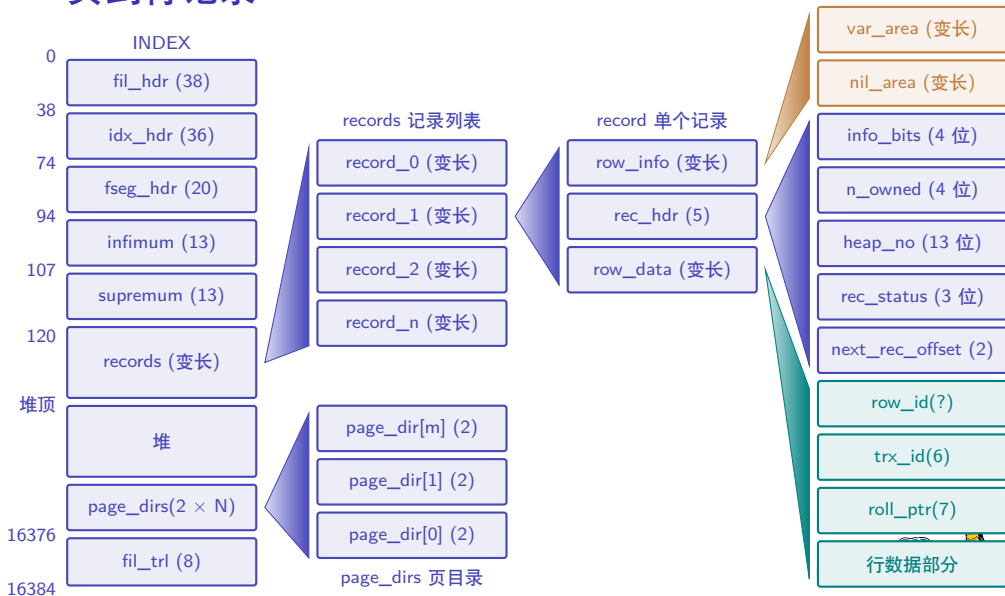


2

COMPACT 行记录



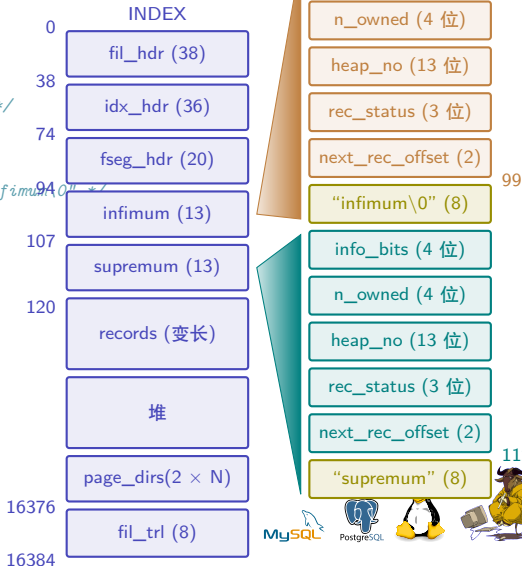
从 Index 页到行记录



infimum 和 supremum

storage/innobase/include/page0page.h

```
79  /** Extra bytes of an infimum record */
80  static const byte infimum_extra[] = {
81      0x01,          /* info_bits=0, n_owned=1 */
82      0x00, 0x02     /* heap_no=0, status=2 */
83      /* ?, ?      */ /* next=(first user rec, or supremum) */
84  };
85  /** Data bytes of an infimum record */
86  static const byte infimum_data[] = {
87      0x69, 0x6e, 0x66, 0x69, 0x6d, 0x75, 0x6d, 0x00 /* "infimum\0" */
88  };
89  /** Extra bytes and data bytes of a supremum record */
90  static const byte supremum_extra_data[] = {
91      /* 0x0?, */ /* info_bits=0, n_owned=1..8 */
92      0x00,
93      0x0b, /* heap_no=1, status=3 */
94      0x00,
95      0x00, /* next=0 */
96      0x73,
97      0x75,
98      0x70,
99      0x72,
100     0x65,
101     0x6d,
102     0x75,
103     0x6d /* "supremum" */
104  };
```



rec_hdr 字段含义

- 源码 `storage/innobase/rem/rec.h`

- info_bits 总共 4 位, 每位标记一种状态

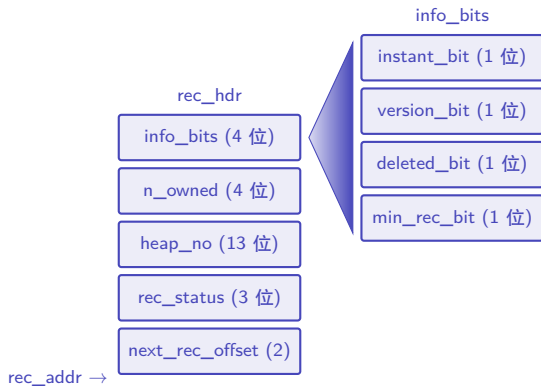
- ▶ MIN_REC_FLAG 标记预先定义的最小记录
- ▶ DELETED_FLAG 标记记录已经删除 `.../rec.h`

```
constexpr uint32_t REC_INFO_MIN_REC_FLAG = 0x10UL;  
constexpr uint32_t REC_INFO_DELETED_FLAG = 0x20UL;  
constexpr uint32_t REC_INFO_VERSION_FLAG = 0x40UL;  
constexpr uint32_t REC_INFO_INSTANT_FLAG = 0x80UL;
```

- n_owned 该记录拥有的记录数
- heap_no 索引堆中该记录的排序记录
- rec_status 记录状态 `.../rec.h`

```
152 /* Record status values */  
153 constexpr uint32_t REC_STATUS_ORDINARY = 0;  
154 constexpr uint32_t REC_STATUS_NODE_PTR = 1;  
155 constexpr uint32_t REC_STATUS_INFIMUM = 2;  
156 constexpr uint32_t REC_STATUS_SUPREMUM = 3;
```

- next_rec_offset 下个记录的偏移
 - ▶ 如果当前记录的地址是 rec_addr
 - ▶ 则下个记录为 rec_addr+next_rec_offset



rec_addr →



row_info 和 row_data

- row_info 记录数据的元信息

- var_area 是变长字段长度

- ① 只有变长字段才有长度记录, 例如 varchar
 - ② 根据字段总长度占有 1 或 2 字节

- nil_area 是字段是否为空标志位

- ① 每一个位标志一个可空字段是否为空
 - ② 默认按照字节对齐, 即不足 8 个按 8 位算

- row_data 记录数据的值

- 前 3 列数据是数据库自动添加的

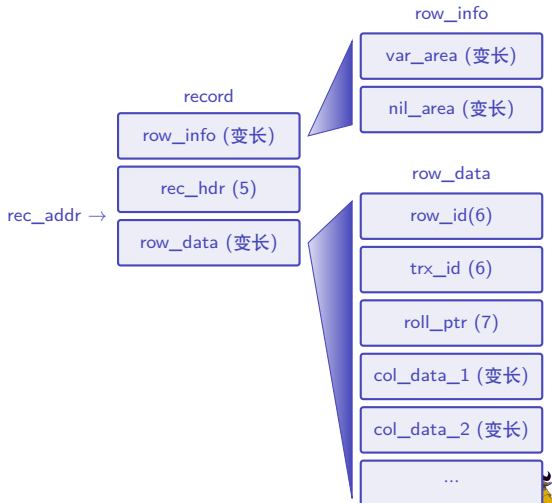
- ① row_id 行数据的唯一 ID, 有时也是 key
 - ② trx_id 事务 ID
 - ③ roll_ptr 回滚指针

- 数据按照表定义的顺序排列

- nil_area 标记为空的列不占字节

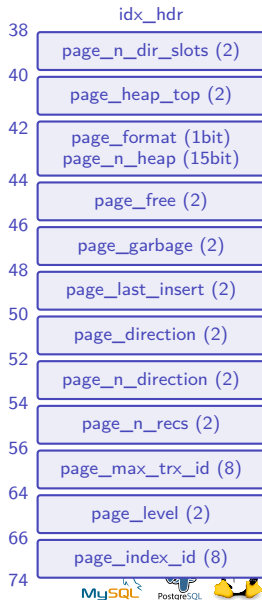
- var_area 变长列先读取长度, 后确定字节

- 定长列占用的字节数更加字段类型确定



回顾上一讲的 idx_hdr

- page_n_dir_slots 页目录槽的数量
- page_heap_top 第一条 Record 位置
- page_n_heap 堆中的 Record 数
 - ▶ 高 1 位被用作 page_format
- REC_FORMAT_REDUNDANT = 0,
- REC_FORMAT_COMPACT = 1,
- page_free 空闲 Record 位置
- page_garbage 被删除的 Record 位置
- page_last_insert 最新插入的 Record
- page_direction 最新插入的 Record 方向
- page_n_direction 相同方向连续插入 Record 数量
- page_n_recs Record 数量
- page_max_trx_id 最大事务 ID
 - ▶ 二级索引和 insert buffer 用的
- page_level 在 B+ 树的深度
- page_index_id 索引 ID



3

行记录实验



实验准备 init-record-list.sql

- 创建测试表 t

- ▶ k 作为主键
- ▶ v 作为数据值

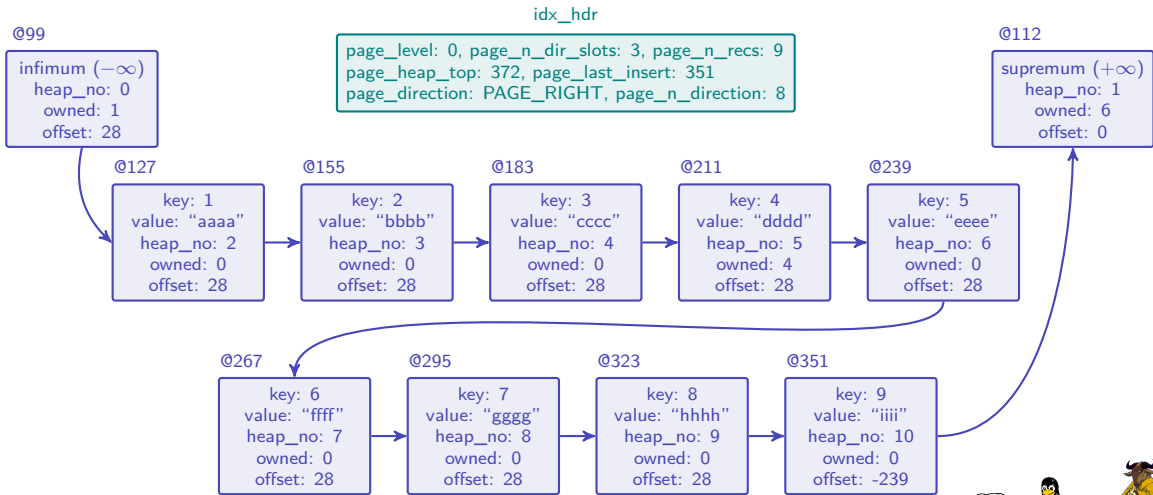
```
create table t (k int primary key, v char(4));
```

- 插入测试数据

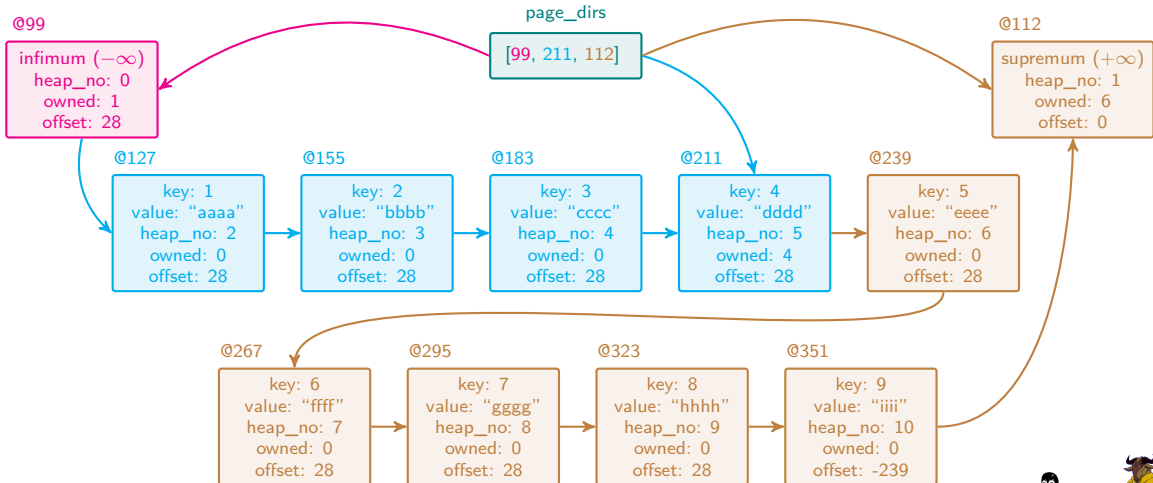
```
insert into t values(1, 'aaaa');  
insert into t values(2, 'bbbb');  
insert into t values(3, 'cccc');  
insert into t values(4, 'dddd');  
insert into t values(5, 'eeee');  
insert into t values(6, 'ffff');  
insert into t values(7, 'gggg');  
insert into t values(8, 'hhhh');  
insert into t values(9, 'iiii');
```



单页面记录链表快照 (Record List)



页目录和 n_owned



4

SDI



SDI 背景知识

- SDI (Serialized Dictionary Information) 是序列化字典信息
 - ▶ 它是 MySQL 8.0 重新设计数据字典后引入的新产物
 - ▶ 用于以序列化的格式存储数据库对象的元数据, 通常是 JSON 格式
- 不同的存储引擎的 SDI 格式有所不同
 - ▶ InnoDB 存储引擎, SDI 数据被存储在表空间的 SDI 页中
 - ▶ 非 InnoDB 存储引擎, 以 `tablename.sdi` 文本文件存储在数据库目录下
- MySQL 8.0 提供了 `ibd2sdi` 工具来提元信息, 命令 `ibd2sdi table01.ibd`

- ▶ 结果中包含, `id` 和 `type` 字段

```
[  
  "ibd2sdi",  
  {"type": 1, "id": 539, "object": {...} },  
  {"type": 2, "id": 70, "object": {...} }  
]
```

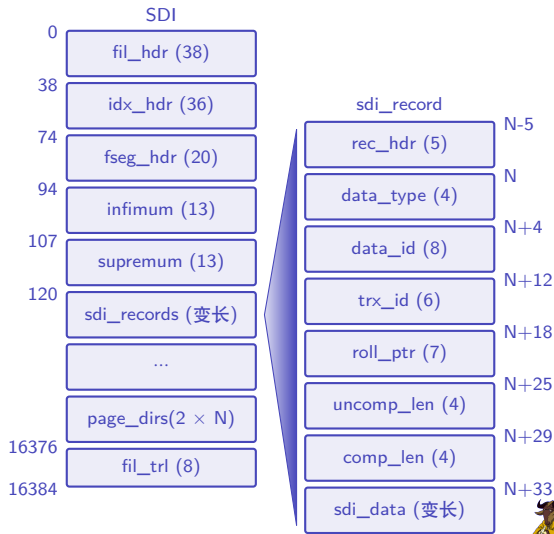
- ▶ `type` 值的含义, 见 `sql/handler.h`

```
119  /** Id for identifying Table SDIs */  
120  constexpr const uint32 SDI_TYPE_TABLE = 1;  
121  
122  /** Id for identifying Tablespace SDIs */  
123  constexpr const uint32 SDI_TYPE_TABLESPACE = 2;
```



SDI 页

- SDI 页和 INDEX 页的格式类似
 - ▶ 只不过把数据记录放在了 records 位置
 - ▶ SDI 数据记录称作 sdi_records
- 数据记录 sdi_records 包含以下几部分
 - ▶ rec_hdr, 和行记录的 rec_hdr 格式相同
 - ▶ sdi_hdr, 记录 SDI 数据的元信息
 - ① data_type 数据类型
 - ② data_id 数据 ID
 - ③ trx_id 事务 ID
 - ④ roll_ptr 回滚指针
 - ⑤ uncomp_len 压缩前长度
 - ⑥ comp_len 压缩后长度
 - ▶ sdi_data 压缩后的数据
 - ① 这部分数据长度为 comp_len
 - ② 使用 zlib 算法压缩
- 独立表空间的 SDI 页通常包含两个对象
 - ▶ table 元信息
 - ▶ tablespace 元信息
 - ▶ 数据和 ibd2sdi 输出一致



聚簇索引

- 首先查找索引定义 indexes
- elements 是索引包含的列

```
"indexes": [  
  {  
    "tablespace_ref": "rtc/t"  
    "name": "PRIMARY",  
    "elements": [  
      {  
        "ordinal_position": 1,  
        "length": 4,  
        "order": 2,  
        "hidden": false,  
        "column_opx": 0  
      },  
      {  
        "ordinal_position": 2,  
        "length": 4294967295,  
        "order": 2,  
        "hidden": true,  
        "column_opx": 2  
      },  
      {  
        "ordinal_position": 3,  
        "length": 4294967295,
```

- columns 包含很多列信息
- 通过 column_opx 字段索引列信息

```
"columns": [  
  {  
    "name": "k",  
    "type": 4,  
    "is_nullable": false,  
    "is_auto_increment": false,  
    "is_virtual": false,  
    "hidden": 1,  
    "ordinal_position": 1,  
    "char_length": 11,  
    "datetime_precision": 0,  
    "datetime_precision_null": 1,  
    "has_no_default": true,  
    "default_value_null": false,  
    "default_value": "AAAAAA==",  
    "options": "interval_count=0;",  
    "se_private_data": "table_id=1127;",  
    "engine_attribute": "",  
    "secondary_engine_attribute": "",  
    "column_key": 2,  
    "column_type_utf8": "int",
```



5

行数据解析



存储格式

- MySQL 不同的数据都是二进制方式存储的，每种类型格式规格如下 ⓘ
- 数组类型
 - ▶ tinyint (1 字节), smallint (2 字节), int / integer (4 字节), bigint (8 字节)
 - ▶ float (4 字节), double (8 字节), real (8 字节)
 - ▶ float(p)
 - ① $0 \leq p \leq 24$ 时 4 字节
 - ② $25 \leq p \leq 53$ 时 8 字节
- 日期类型
 - ▶ year (1 字节), date (3 字节), time (3 字节)
 - ▶ datetime (5 字节), timestamp (4 字节)
- 字符串类型
 - ▶ char(n) 定长字符串, n 字节, 这里 n 最大值为 255
 - ▶ varchar(n) 变长字符串, 记录字符串长度和数据
 - ① 长度记录在 var_area 区域
 - ② 总长度不超过 n 个字符
 - ③ 总字节数需要根据字符集计算, 例如: utf8mb4 每个字符最多 4 字节



解析规则

- int 格式, 分为符号位 signed 和数值位 value



- ▶ 如果 signed=1, 则当前数值为 (int) (value & 0x7fffffff)
- ▶ 如果 signed=0, 则当前数值为 (int) (value | 0x80000000)


- date 格式, 年月日分不同位存储



- datetime 格式, MySQL 8.0 中使用 5 字节紧凑格式存储



- ▶ month = year_month % 13
- ▶ year = year_month / 13
- ▶ 其他字段就是对应位的数值

- char / varchar 格式见官网行格式说明 



结束

