[MYSQL] row_format=compressed的存储结构浅析

原创 大大刺猬 2025-07-18 54

导读

我们之前已经介绍了3种row_format格式:REDUNDANT,COMPACT,DYNAMIC. 现在来讲最后一种:COMPRESSED

有的小伙伴可能会疑惑之前不是讲过压缩吗? 就那个zlib和lz4那俩啊. 那俩是PAGE级别的压缩,除了FSP的'page'都压; 今天讲的是行级别的压缩,只压缩'行'.

行压缩的结构

行格式为压缩的表的创建方式

```
-- 建表时指定为压缩行
create table t20250718_1(id int, c1 varchar(200)) row_format=compressed;

-- 建表后修改为压缩表
alter table t20250718_1 row_format=compressed;

-- 设置页大小为4K
create table db1.t20250718_2(id int) row_format=compressed KEY_BLOCK_SIZE=4;
```

```
Database changed
(root@127.0.0.1) [db1]> -- * * * * * * * * *
(root@127.0.0.1) [db1]> create table t20250718_1(id int, c1 varchar(200)) row_format=compressed;
compressed KEY_BLOCK_SIZE=4;Query OK, 0 rows affected (0.01 sec)

(root@127.0.0.1) [db1]>
(root@127.0.0.1) [db1]> -- * * * * * * * * *
(root@127.0.0.1) [db1]> alter table t20250718_1 row_format=compressed;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

(root@127.0.0.1) [db1]>
(root@127.0.0.1) [db1]> -- * * * * * * 4K
(root@127.0.0.1) [db1]> create table db1.t20250718_2(id int) row_format=compressed KEY_BLOCK_SIZE=4;
Query OK, 0 rows affected (0.02 sec)

(root@127.0.0.1) [db1]> ^DBye
10:57:28 [root@ddcw21 ~]#ll /data/mysql_3314/mysqldata/db1/t20250718_*.ibd -h
-rw-r----- 1 mysql mysql 56K Jul 18 10:57 /data/mysql_3314/mysqldata/db1/t20250718_1.ibd
-rw-r----- 1 mysql mysql 28K Jul 18 10:57 /data/mysql_3314/mysqldata/db1/t20250718_2.ibd
10:57:29 [root@ddcw21 ~]#
```

很简单, 就是执行 `row_format=compressed` .

> 我们知道Innodb会对表初始分配7个page, 8K的page大小就为56K, 4K的page大小为28K, 所以对于16384的page压缩默认是8K

既然叫行的压缩, 那压缩的肯定就算数据行, 也就是只对 `FIL_PAGE_INDEX` 有效. 诶, FIL_PAGE_SDI 也算是FIL_PAGE_INDEX的变种啊, 相当于固定结构的表而已.

既然表元数据信息都可能是压缩的了, 那问题来了, 怎么确定这个表是压缩的呢?
总不能一点点猜吧...

确认表的压缩大小

既然元数据信息被压缩了, 那我们就找更元的元数据信息-FSP, 这个页不会被加密和压缩的. 其中有个叫FSP_SPACE_FLAGS的东西, 如下图::

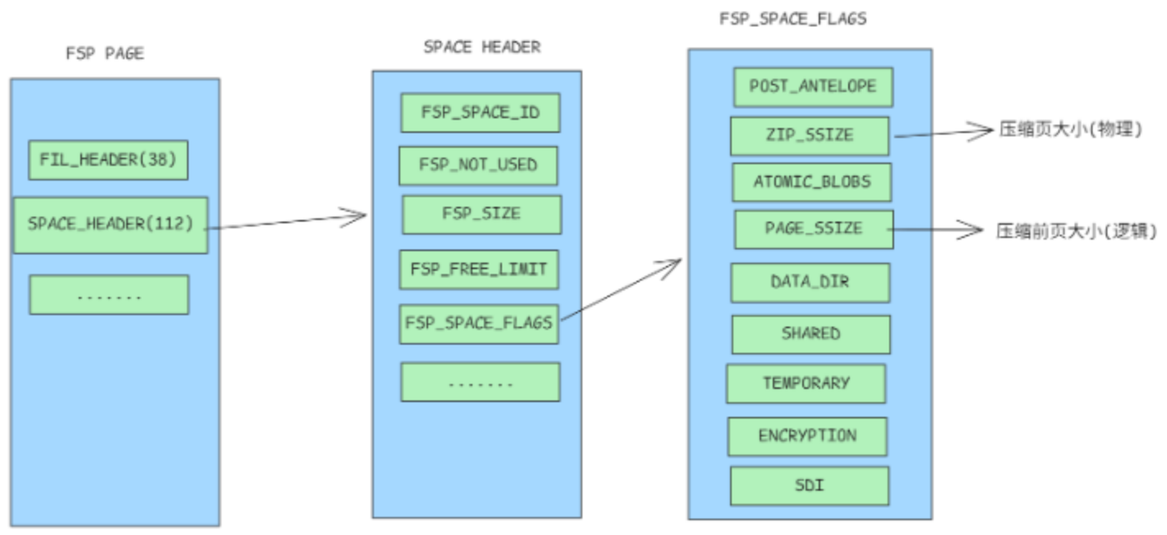
大大刺猬
关注
146 文章 108 粉丝 66K+ 浏览量
获得了 350 次点赞
内容获得 70 次评论
获得了 257 次收藏

TA的专栏
PYTHON解析MYSQL
收录 59 篇内容

热门文章
MYSQL 文件解析 (3) 数据文件(ibd)解析 2023-04-23 2801浏览
ibd2sql解析ibd文件为SQL 2023-04-27 2060浏览
[MYSQL] 数据恢复, 无备份, 只剩一个 ibd 文件 怎么恢复数据? 2024-04-12 1948浏览
MYSQL 文件解析 (1) binlog 文件解析 2023-04-23 1707浏览
mysql-5.7.38启动流程源码解读 2022-09-26 1529浏览

在线实训环境入口
MySQL在线实训环境 查看详情 »

最新文章
[MYSQL] 修改字段长度的时候不能使用instant算法? 其实inplace就够了 2025-07-30 49浏览
[MYSQL] 修改字段长度的时候不能使用instant算法? 那就定制一个? 2025-07-25 81浏览
[MYSQL] 备份失败,但是啥日志信息都没有 2025-07-15 72浏览
[MYSQL] 从库 io_thread 接受binlog速度太慢? 2025-07-11 326浏览
[MYSQL] 参数/变量浅析(1) -- 超时(timeout)相关 2025-07-03 121浏览



对于计算方式可参考如下py代码(include/fsp0types.h):

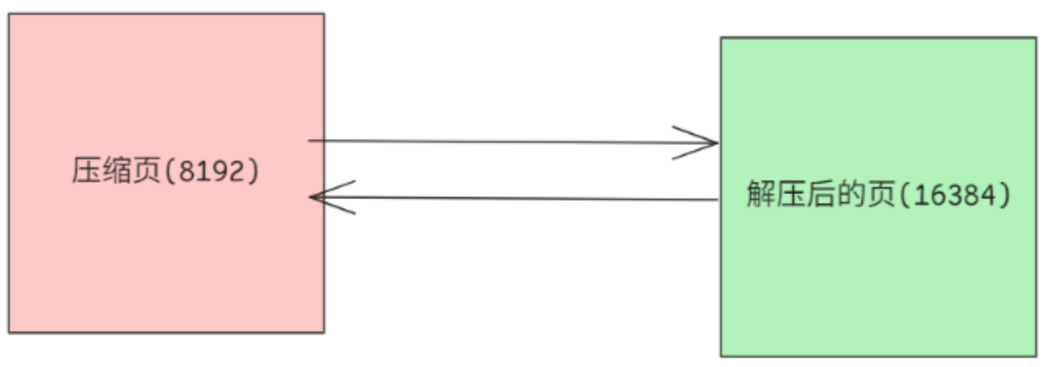
```
def GET_FSP_STATUS_FROM_FLAGS(flags):
        logical_size = 16384 if ((flags & 960) >> 6) == 0 else 512 << ((flags & 960) >> 6)
        physical_size = logical_size if ((flags & 30) >> 1) == 0 else 512<<((flags & 30) >>
        compressed = False if ((flags & 30) >> 1) == 0 else True
        return {
                'POST_ANTELOPE':(flags & 1) >> 0,
                'ZIP_SSIZE':(flags & 30) >> 1,
                'ATOMIC_BLOBS':(flags & 32) >> 5,
                'PAGE_SSIZE':(flags & 960) >> 6,
                'DATA_DIR':(flags & 1024) >> 10,
                'SHARED':(flags & 2048) >> 11,
                'TEMPORARY':(flags & 4096) >> 12,
                'ENCRYPTION':(flags & 8192) >> 13,
                'SDI':(flags & 16384) >> 14,
                'logical_size':logical_size, # logical page size (in memory)
                'physical_size':physical_size, # physical page size (in disk)
                'compressed':compressed
        }
```

这里的ZIP_SSIZE就是实际上在磁盘上存储的大小, 使用4bit表示(512一个block). 比如值为4时, 物理块大小就为: `512<<4` (8192), 即磁盘上的块大小为8192.

PAGE_SSIZE就是逻辑大小, 也就是innodb_page_size. 数据解析的时候, 我们先要把压缩后的大小(ZIP_SSIZE)解压到压缩前的大小(PAGE_SSIZE) 然后就可以当作普通页(DYNAMIC)来处理了.

压缩行结构

如果每次数据更新我们都要压缩和解压的话, 成本有点高啊.



所以应该只压缩其中的一部分数据,
新insert进来的, 就直接放压缩数据后面, 待空间不够之后再去压缩.
而要删除的数据也是应该只打标记即可. 于是我们翻阅源码(page_zip_decompress_low)后得到如下结构:

目录

- 导读
- 行压缩的结构
  - 确认表的压缩大小

对象	大小	描述
FIL_HEADER+PAGE_HEADER	94	页基础信息
compressed_data	x	压缩的数据
uncompressed_data	y	未压缩部分的数据
...		未使用的空间
overflow page	20*m	溢出页的记录信息, 还是每条20字节
trx_id+rollptr	13*n	事务和回滚指针相关信息
page diretory	2*n	page dir信息 (基于压缩前的页)

看起来合情合理, 那我们简单的验证下呢.

验证压缩行结构

我们就没必要把页大小还原回去了. 我们直接开解:

先准备测试数据

```
create table db1.t20250718_compressed(c1 int, c2 varchar(20), c3 text) row_format=compresse
insert into  db1.t20250718_compressed values(1,'xx','yy');
insert into  db1.t20250718_compressed values(1,'zz',repeat('a',10000));
```

然后使用python来打开ibd文件并解析. 先看下fsp中的FSP_SPACE_FLAGS的相关信息

```
# 声明 GET_FSP_STATUS_FROM_FLAGS 略(见上文)
import struct,zlib
f = open('/data/mysql_3314/mysqldata/db1/t20250718_compressed.ibd','rb')
data = f.read(38+112)
FSP_SPACE_FLAGS = struct.unpack('>L',data[38+16:][:4])[0]
print(GET_FSP_STATUS_FROM_FLAGS(FSP_SPACE_FLAGS))
```

```
...                 'logical_size':logical_size, # logical page size (in memory)
...                 'physical_size':physical_size, # physical page size (in disk)
...                 'compressed':compressed
...         }
...
>>> import struct,zlib
>>> f = open('/data/mysql_3314/mysqldata/db1/t20250718_compressed.ibd','rb')
>>> data = f.read(38+112)
>>> FSP_SPACE_FLAGS = struct.unpack('>L',data[38+16:][:4])[0]
>>> print(GET_FSP_STATUS_FROM_FLAGS(FSP_SPACE_FLAGS))
{'POST_ANTELOPE': 1, 'ZIP_SSIZE': 4, 'ATOMIC_BLOBS': 1, 'PAGE_SSIZE': 0, 'DATA_DIR': 0, 'SHARED': 0, 'TEMPORARY': 0, 'ENCRYPTI
ON': 0, 'SDI': 1, 'logical_size': 16384, 'physical_size': 8192, 'compressed': True}
>>>
```

我们可以看到:物理页大小是8K, 逻辑页大小是16K, 存在sdi信息...

然后我们来解析数据, 由于我们不知道压缩的数据大小, 所以我们得基于流来解析,故使用zlib.decompress obj

```
f.seek(8192*4,0)
data = f.read(8192)
d = zlib.decompressobj()
c = d.decompress(data[94:])
# 初始化基础页信息
unpage = data[:94]
# infimum & supremum
unpage += struct.pack('>BBB',0x01,0x00,0x02)
unpage += data[-2:]
unpage += struct.pack('>8B',0x69, 0x6e, 0x66, 0x69, 0x6d, 0x75, 0x6d, 0x00)
unpage += b'\x03'
unpage += struct.pack('>12B',0x00,0x0b,0x00,0x00,0x73,0x75,0x70,0x72,0x65,0x6d,0x75,0x6d)
# 加上压缩页信息
unpage += c
# 加上未压缩部分的信息
unpage += d.unused_data

# 解析page dir信息:
n_dense = struct.unpack('>H',data[42:44])[0] & 32767
page_dir = struct.unpack(f'>{n_dense-2}H',data[-(2*(n_dense-2)):])
print(page_dir )
```

```
>>> unpage += b'\x03'
>>> unpage += struct.pack('>12B',0x00,0x0b,0x00,0x00,0x73,0x75,0x70,0x72,0x65,0x6d,
>>> unpage += c
>>> unpage += d.unused_data
>>>
>>> # page dir
>>> n_dense = struct.unpack('>H',data[42:44])[0] & 32767
>>> page_dir = struct.unpack(f'>{n_dense-2}H',data[-(2*(n_dense-2)):])
>>> print(page_dir )
(164, 128)
>>>
>>>
```

看到了熟悉的offset, 这里的page dir是记录的所有行(不含infimum & supremum)的偏移量. 然后我们来解析trxid和rollptr

```
# trxid&rollptr
trxid_rollptr_data = data[-(2*rows+13*rows):-(2*rows)]
for i in range(rows):
        print(int.from_bytes(trxid_rollptr_data[i*13:][:6],'big'),int.from_bytes(trxid_roll
```

```
>>> trxid_rollptr_data = data[-(2*rows+13*rows):-(2*rows)]
>>> for i in range(rows):
...     print(int.from_bytes(trxid_rollptr_data[i*13:][:6],'big'),int.from_bytes(trxid_rollptr_data[i*13:][6:13],'big'))
...
75101095 36310272012845328
75101092 36310272012779792
>>>
```

这部分信息其实没多大用...

然后我们再来看数据

```
f.seek(8192*4,0)
data = f.read(8192)
d = zlib.decompressobj()
c = d.decompress(data[94:])
# 初始化基础页信息
unpage = data[:94]
# infimum & supremum
unpage += struct.pack('>BBB',0x01,0x00,0x02)
unpage += data[-2:]
unpage += struct.pack('>8B',0x69, 0x6e, 0x66, 0x69, 0x6d, 0x75, 0x6d, 0x00)
unpage += b'\x03'
unpage += struct.pack('>12B',0x00,0x0b,0x00,0x00,0x73,0x75,0x70,0x72,0x65,0x6d,0x75,0x6d)
# 加上压缩页信息
unpage += c
# 加上未压缩部分的信息
unpage += d.unused_data
```

```
>>> unpage[128:200]
b'\x02\x02\x00\x00%\xe5i\r\x80\x00\x00\x01xyy\x04\x00\x02\xc0\x14\x00\x00%\xe5i\x0e\x80\x00\x00\x01zz\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00'
>>>
>>> c
b'\r\x1b\x08\x00~\x01'
>>>
```

rowid 第一个字段 第2,3字段

我们的压缩数据部分为啥只有个 `b'\r\x1b\x08\x00~\x01'` 呢? 这也不是我们的数据啊!

前面有讲: Insert部分数据会先先非压缩部分, 不然每次都解压又压缩的, 成本老高了.

那这6字节是啥呢? 你猜(提示:可以更改表结构,数据类型来观察其变化)

如果此时我们重新设置表 `row_format=compressed` ,则会将之前未压缩部分的数据进行压缩.

```
alter table db1.t20250718_compressed row_format=compressed;
```

```
>>> data = f.read(8192)
>>> d = zlib.decompressobj()
>>> c = d.decompress(data[94:])
>>> # 初始化基础页信息
>>> unpage = data[:94]
>>> # infimum & supremum
>>> unpage += struct.pack('>BBB',0x01,0x00,0x02)
>>> unpage += data[-2:]
>>> unpage += struct.pack('>8B',0x69, 0x6e, 0x66, 0x69, 0x6d, 0x75, 0x6d, 0x00)
>>> unpage += b'\x03'
>>> unpage += struct.pack('>12B',0x00,0x0b,0x00,0x00,0x73,0x75,0x70,0x72,0x65,0x6d,0x75,0x6d)
>>> # 加上压缩页信息
>>> unpage += c
>>> # 加上未压缩部分的信息
>>> unpage += d.unused_data
>>>
>>> c
b'\r\x1b\x08\x00~\x01\x02\x02\x00\x00\x00%\xe5i\r\x80\x00\x00\x01xyy\x14\xc0\x02\x00\x00\x00%\xe5i\x0e\x80\x00\x00\x01zz'
>>>
>>>
```

细心的小伙伴可能会发现压缩前每行数据前面好像有个 ID/NO 之类的东西, 但是压缩进来后就没了

溢出页我们就不看了, 格式是和DYNAMIC一样的, 只是存储的位置换到了rollptr位置处.

总结

mysql有2种压缩方式, 1种是基于行的(row_format=compressed),另一种是基于page的(compression=zlib/lz4); 后者需要OS的文件系统支持才行. 当然我们的ibd2sql下个版本也会支持这种格式的.

```
18:07:31 [root@ddcw21 ibd2sql_v2.x]#python3 main.py /data/mysql_3314/mysqldata/db1/t20250718_compressed.ibd --ddl
CREATE TABLE IF NOT EXISTS `db1`.`t20250718_compressed` (
  `c1` int DEFAULT NULL,
  `c2` varchar(20) DEFAULT NULL,
  `c3` text
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci ROW_FORMAT=COMPRESSED;
18:07:44 [root@ddcw21 ibd2sql_v2.x]#
```

参考:

https://dev.mysql.com/doc/refman/8.0/en/innodb-row-format.html

https://github.com/mysql/mysql-server/tree/trunk/storage/innobase

墨力计划 mysql

文章被以下合辑收录

PYTHON解析MYSQL（共59篇）

用python解析mysql协议,模拟mysql连接,解析binlog,redolog,ibd文件等

收藏合辑

评论

分享你的看法，一起交流吧～

相关阅读

ACDU周度精选 | 本周数据库圈热点 + 技术干货分享（2025/7/25期）
墨天轮小助手 469次阅读 2025-07-25 15:54:18

ACDU周度精选 | 本周数据库圈热点 + 技术干货分享（2025/7/17期）
墨天轮小助手 436次阅读 2025-07-17 15:31:18

墨天轮「实操看我的」数据库主题征文活动启动
墨天轮编辑部 379次阅读 2025-07-22 16:11:27

深度解析MySQL的半连接转换
听见风的声音 204次阅读 2025-07-14 10:23:00

MySQL 9.4.0 正式发布，支持 RHEL 10 和 Oracle Linux 10
严少安 199次阅读 2025-07-23 01:21:32

索引条件下推和分区-一条SQL语句执行计划的分析
听见风的声音 196次阅读 2025-07-23 09:22:58

null和子查询--not in和not exists怎么选择？
听见风的声音 182次阅读 2025-07-21 08:54:19

MySQL数据库SQL优化案例(走错索引)
陈举超 164次阅读 2025-07-17 21:24:40

使用 MySQL Clone 插件为MGR集群添加节点
黄山谷 162次阅读 2025-07-23 22:04:19

MySQL 8.0.40：字符集革命、窗口函数效能与DDL原子性实践
shunwah 140次阅读 2025-07-15 15:27:19