MySQL学习笔记（Day023：InnoDB_4 – 页(2)/行记录）

MySQL学习

一. 主键实验

1. 多个唯一非空键

```
mysql> create table test_key (
    -> a int,
    -> b int not null,
    -> c int not null,
    -> unique key(a),
    -> unique key(c),
    -> unique key(b)
    -> );
Query OK, 0 rows affected (0.16 sec)

mysql> insert into test_key values(1,2,3),(4,5,6),(7,8,9);
Query OK, 3 rows affected (0.04 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from test_key;
+------+---+---+
| a    | b | c |
+------+---+---+
|    1 | 2 | 3 |
|    4 | 5 | 6 |
|    7 | 8 | 9 |
+------+---+---+
3 rows in set (0.00 sec)

mysql> select *, _rowid from test_key; -- _rowid是主键值
+------+---+---+--------+
| a    | b | c | _rowid |
+------+---+---+--------+
|    1 | 2 | 3 |      3 |  -- 可以发现，这里的主键是c
|    4 | 5 | 6 |      6 |
|    7 | 8 | 9 |      9 |
+------+---+---+--------+
3 rows in set (0.00 sec)

mysql> create table test_key_2 (
    -> a varchar(8),  -- 使用varchar类型
    -> b varchar(8) not null,
    -> c varchar(8) not null,
    -> unique key(a),
    -> unique key(c),
    -> unique key(b)
    -> );
Query OK, 0 rows affected (0.15 sec)

mysql> insert into test_key_2 values('a','b','c'),('d','e','f'),('g','h','i');
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from test_key_2;
+------+---+---+
| a    | b | c |
+------+---+---+
| a    | b | c |
| d    | e | f |
| g    | h | i |
+------+---+---+
3 rows in set (0.00 sec)

mysql> select *, _rowid from test_key_2;
ERROR 1054 (42S22): Unknown column '_rowid' in 'field list'   -- 报错了
-- _rowid只能是在key的类型为整型时才有效

-- 方法一
mysql> select * from information_schema.columns where table_name="test_key_2" and column_key="PRI"\G
*************************** 1. row ***************************
           TABLE_CATALOG: def
            TABLE_SCHEMA: burn_test
              TABLE_NAME: test_key_2
             COLUMN_NAME: c   -- 该列的列名是 c
        ORDINAL_POSITION: 3
          COLUMN_DEFAULT: NULL
             IS_NULLABLE: NO
               DATA_TYPE: varchar
CHARACTER_MAXIMUM_LENGTH: 8
  CHARACTER_OCTET_LENGTH: 32
       NUMERIC_PRECISION: NULL
           NUMERIC_SCALE: NULL
      DATETIME_PRECISION: NULL
      CHARACTER_SET_NAME: utf8mb4
          COLLATION_NAME: utf8mb4_general_ci
             COLUMN_TYPE: varchar(8)
              COLUMN_KEY: PRI  -- 该列是主键
                   EXTRA:
              PRIVILEGES: select,insert,update,references
          COLUMN_COMMENT:
    GENERATION_EXPRESSION:
1 row in set (0.00 sec)

-- 方法二
mysql> desc test_key_2;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| a     | varchar(8) | YES  | UNI | NULL    |       |
| b     | varchar(8) | NO   | UNI | NULL    |       |
| c     | varchar(8) | NO   | PRI | NULL    |       | -- key 是PRI ，就可以知道 c 列是主键
+-------+------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

2. 系统定义主键（系统rowid）

当用户表中没有显示的指定主键，且没有非空唯一键时，系统会 自定义 一个 主键（**6个字节，int型，全局，隐藏**）

```
mysql> create table test_key_3(
    -> a int,
    -> b int,
    -> c int);
Query OK, 0 rows affected (0.11 sec)

mysql> insert into test_key_3 values(1,2,3),(4,5,6),(7,8,9);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select *,_rowid from test_key_3;
ERROR 1054 (42S22): Unknown column '_rowid' in 'field list'  -- 这里无法用_rowid查看，因为系统rowid对用户是透明的
```

> 假设有 a 和 b 两张表都使用了系统定义的主键，则系统定义的主键的ID 不是 在 表内 进行 单独递增 的，而是 全局递增 。
> 该系统的rowid是定义在 `ibdata1.ibd` 中的 `sys_rowid` 中，全局自增
> 6个字节表示的数据量为 2^48 ，通常意义上是够用的

注意：强烈建议自己显示定义主键

二. 页的结构

1. File Header

名称	大小	备注

名称	大小	备注
FIL_PAGE_SPACE_OR_CHKSUM	4	
FIL_PAGE_OFFSET	4	
FIL_PAGE_PREV	4	其实是page_number (前一个)
FIL_PAGE_NEXT	4	其实是page_number (后一个)
FIL_PAGE_LSN	8	
FIL_PAGE_TYPE	2	
FIL_PAGE_FILE_FLUSH_LSN	8	
FIL_PAGE_ARCH_LOG_NO_OR_SPACE_ID	4	

> 在一个页中不仅仅只有记录，还有 `File Header`，`Page Header`，`File Trailer` 等

三. 记录

1. ROW_FORMAT

- **REDUDENT**： 兼容老版本的InnoDB，MySQL 4.1版本之前
- **COMPACT**： `MySQL 5.6` 版本的默认格式
- **COMPRESSED**：支持压缩
- **DYNAMIC**：大对象记录优化， `MySQL 5.7` 版本默认格式

2. COMPACT 结构

- **variable string length list**
  - 变长字段 列表，表示有 多少个 变长字段，且序号 逆序 显示
- **NULL flag**
  - 是否有NULL值
- **rowid**
  - B+树索引键值
- **trx id**
  - 事物ID，6个字节
- **roll pointer**
  - 回滚指针，7个字节

3. COMPACT 示例

1. 创建mytest表，格式为compact，且没有显示定义主键和非空唯一键，故使用系统定义的ROWID。并插入三条记录。

```
create table mytest (
   t1 varchar(10),
   t2 varchar(10),
   t3 char(10),
   t4 varchar(10))
engine=innodb row_format=compact;
insert into mytest values ('a','bb','bb','ccc');
insert into mytest values ('d','ee','ee','fff');
insert into mytest values ('d',NULL,NULL,'fff');
```

2. 将mytest表结构进行dump
   图中红色部分对应第一条记录，黄色部分对应第二条记录，深蓝色部分对应第三条记录

```
0000c070  73 75 70 72 65 6d 6d 03 02 01 00 00 00 10 00  |supremum........|
0000c080  2c 00 00 00 2b 68 00 00  00 00 00 06 05 80 00 00  |,...+h..........|
0000c090  00 32 01 10 61 62 62 62  62 20 20 20 20 20 20 20  |.2..abbbb      |
0000c0a0  20 63 63 63 03 02 01 00  00 00 18 00 2b 00 00 00  | ccc........+...|
0000c0b0  2b 68 01 00 00 00 00 06  06 80 00 00 00 32 01 10  |+h...........2..|
0000c0c0  64 65 65 65 65 20 20 20  20 20 20 20 20 66 66 66  |deeeefff       |
0000c0d0  03 01 06 00 00 20 ff 98  00 00 00 2b 68 02 00 00  |..... .....+h...|
0000c0e0  00 00 06 07 80 00 00 00  32 01 10 64 66 66 66 00  |........2..dfff.|
```

3. 将红色部分对应的第一条记录进行解析

```
03 02 01/*reverse*/
00 /*NULL flag*/
00 00 10 00 2c /*Record Header, 5 bytes*/
00 00 00 2b 68 00/*RowID */
00 00 00 00 06 05/*TransactionID*/
80 00 00 00 32 01 10/*Roll Pointer*/
61/*col1'a'*/
62 62/*col2 'bb'*/
62 62 20 20 20 20 20 20 20 20/*col3'bb' */
63 63 63/*col4'ccc'*/
```

   - **variable string length list**
     - 03 02 01 ，表示有三个变长字段（ varchar，varbinary，text等 ），且逆序存放（ 为了提高CPU的cache的命中率 ）
   - **NULL flag**
     - 00 , 这条记录中不存在NULL
   - **Record Header**
     - 5个字节，比较底层（ 比如看该记录有没有被删除 ）
   - **`RowID`**
     - 主键ID, 00 00 00 2b 68 00，从这个值可以看出，不是每张表从1开始递增的，是全局的ROWID
   - **`TransactionID`**
     - 事物ID
   - **`Roll Pointer`**
     - 回滚指针

3.1. char 和 varchar 的区别

详细的操作可以参看之前的笔记 **Day008：数据类型**

> **结果：**
> 在 多字节字符集（ 如UTF8mb4 ）下：
>
> 1. `char(N)` 中存储的数据的长度 范围 是 `N ~ 4N` ，当存储数据的长度 `M` ， 未达到`N` 时，则填充空格（ 0x20 ）,且空格的长度 取最小 的长度 `N-M` ，而 不是`4N-M`
> 2. `varchar(N)` 则 不填充空格
>
> **注意： `char` 数据类型本来是定长数据，但是在 多字节字符集 下，表现的行为和 `varchar` 类似，失去了原来的优势，当数据更新变长后可能无法 原地更新**

3.2. 原地更新（ in place update ）

```
+-------+-------------------+---------+-------+-------------+
|       |                   |         |       |             |
| Data1 |  Data2 len = 10   |  Data3  | ......|     Free    |
|       |                   |         |       |    Space    |
+-----------------------+---------+-------+-------------+
        |
        |
        |   Data2 update +->  DataA len=10   (in place update)
        |
        v
+-------------------------+---------+-------+-------------+
|       |                 |         |       |             |
| Data1 |  DataA len = 10 |  Data3  | ......|     Free    |
|       |                 |         |       |    Space    |
+-------------------------+---------+-------+-------------+
        |
        |
        |   DataA update +->  DataB len=16
        |
        v
+-----------------------+--------+------+---------+---------+
|       |               |        |      |         |         |
| Data1 |Add this space | Data3  |......| DataB   |   Free  |
|       |to Free_List   |        |      | len=16  |  Space  |
+-----------------------+--------+------+---------+---------+
```

- 原地更新不会占用新的存储空间
- 非原地更新需要删除（ 物理删除 ）原来的空间的数据，然后将更新后的数据插入到页的后面
- 删除的数据的空间，会插入到 `Free_List` 链表的 头部
- 原地更新 不会触发页的分裂

> `Free_List` 是将页中被删除的空间串联在一起（ 组成一个 链表 ），当有数据被插到页内时，先看一下Free_list中 第一个空间 的大小，如果 空间合适 ，就将该记录 插入 到 第一个空间 中去，如果 不合适 ，直接插入到 页的尾部 的剩余空间。（ 不会去看`Free_list`的第二个空间 ）
>
> 当该页的数据被插满了，不会马上进行分页，而是进行 **`reorganize`** 操作，即将页内的数据在 内存 中进行整理，然后覆盖原来的页（ 不影响性能 ），所以InnoDB 不需要碎片整理 。

3.3 Reorganize

```
+--------+--------+--------+                 +--------+--------+--------+
|        |        |        |                 |        |        |        |
|  Data  |        |        |                 |  Data  |  Data  |  Data  |
|        |        |        |                 |        |        |        |
+-------------------------+                 +-------------------------+
|        |        |        |                 |        |        |        |
|  Data  |        |  Data  |   Reorganize    |  Data  |  Data  |        |
|        |        |        |  +------------> |        |        |        |
+-------------------------+                 +-------------------------+
|        |        |        |                 |        |        |        |
|        |  Data  |  Data  |                 |        |        |        |
|        |        |        |                 |        |        |        |
+--------+--------+--------+                 +--------+--------+--------+
```

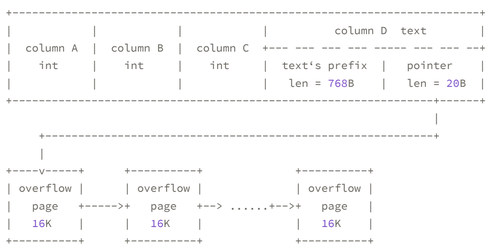
4. DYNAMIC

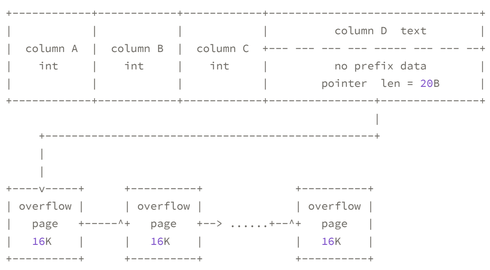
> 官方文档

DYNAMIC相比COMPACT，优化了大对象记录的存储。

假设有一条记录有A，B，C，D 四列，其中D列的是text类型，且含有2W个字节的长度。

- **COMPACT**
  COMPACT会存储text中的前768个字节的数据，剩余的数据通过20个字节的指针指向溢出页





> 相对COMPACT，DYNAMIC在一个页中存储的 记录数 更多（*因为有768字节的prefix，一条记录的字节假设是800字节，那16K的页只能存放20条记录，而之前我们测算可以存放80条记录*），这样一来，B+树的高度可能会变高，读取的IO次数可能会变多。
>
> 一个页能存放的记录越多，则性能越优

四. 编译安装MySQL

```
# 预备工作
shell> yum install gcc glibc gcc-c++ ncurses-devel bison

# 下载安装
shell> wget http://cdn.mysql.com//Downloads/MySQL-5.7/mysql-5.7.10.tar.gz
shell> wget http://cdn.mysql.com//Downloads/MySQL-5.7/mysql-boost-5.7.10.tar.gz


shell> tar zxf mysql-5.7.10.tar.gz
shell> tar zxvf mysql-boost-5.7.10.tar.gz
shell> cd mysql-5.7.10
shell> mkdir mybuild
shell> cmake .. -DWITH_BOOST=../boost/
shell> make  # or make -j cpu数目
shell> make package  # 生成了mysql-5.7.10-linux-x86_64.tar.gz
                     # 后续就是把他当作一个二进制包，和之前一样进行安装即可

# 安装完成后，启动MySQL，然后检查err.log
......
[Note] InnoDB: PUNCH HOLE support available # 说明已经支持透明压缩功能
......
```