MySQL 8.0 INSTANT DDL 算法原理简析

原创 鸟山明 丹柿小院 2025年04月03日 10:19 陕西

引言

MySQL 8.0 中引入 INSTANT DDL，极大提高了 DDL 的执行效率，具体原理简单说就是仅修改元数据，不修改现有行的物理结构，显然这样会导致数据与表结构不一致，这个问题是怎么解决的呢？本文将简单分析这个问题，并先回顾 online DDL。

概念

表结构

MySQL 属于关系型数据库，关系型数据库的基本单位是表（或称为关系），每个表由行（记录）和列（字段）组成，具体由表结构（schema）定义表的组织方式，包括表中可以存储哪些数据以及这些数据的类型。

MySQL 属于磁盘数据库，数据最终保存磁盘上，包括表结构与数据。

MySQL 5.7 中每个表对应两个文件。

```
[root@DB test_zk]# ll -h t2.*
-rw-r----- 1 mysql mysql 8.5K Dec 18 18:28 t2.frm
-rw-r----- 1 mysql mysql 112K Dec 18 18:28 t2.ibd
```

其中：

- frm 文件，保存表的元数据，包括表的定义（如列名、数据类型、索引等），显然**对于一张表的每行数据，表结构都是相同的，因此如果每行数据单独保存表结构，将导致空间浪费**；
- ibd 文件，保存数据和索引，前提是开启独立表空间。

MySQL 8.0 中每个表对应一个文件。

```
[root@DB test_zk]# ll -h t2*
-rw-r----- 1 mysql mysql 128K Feb  4  2024 t2.ibd
```

其中：

- 不包括 frm 文件，原因是将 frm 文件中的元数据从二进制文本格式转移到了事务型数据字典中；
- 将表结构保存在 InnoDB 系统表的优点是确保元数据操作的原子性和崩溃恢复能力。

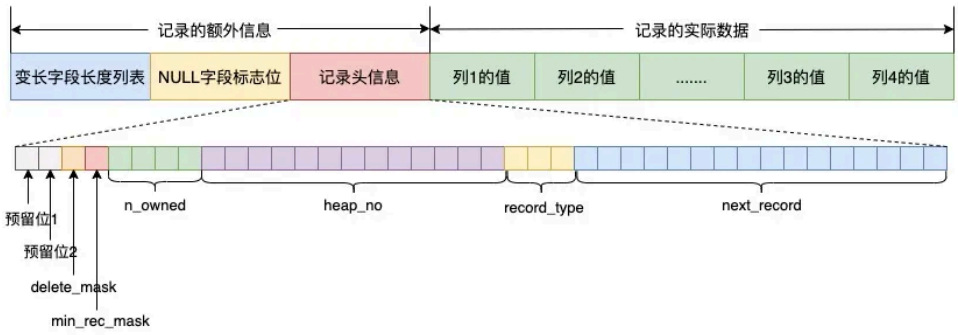
MySQL 内存 buffer pool 中保存数据页，同样表结构也会保存在内存中，从而支持快速查询和操作。

内存中表结构的使用场景：

- 第一次访问表时，将元数据从磁盘加载到内存中，MySQL 为每一张表创建一个 `TABLE_SHARE` 对象，其中保存表结构的全局共享信息（如字段定义、索引结构）；
- 对于查询操作，SQL 解析时，从内存中的 `TABLE_SHARE` 获取表结构，验证列名、类型、索引是否存在；
- 对于更新操作，DDL 执行期间更同步更新磁盘与内存中的元数据，MySQL 8.0 中可以保证 DDL 的原子性。

行格式

MySQL 中数据与表结构分开存储，每行数据通过行格式进行组织，如下图所示是 COMPACT 行格式。



其中：

- info_bits 是记录头信息中的一部分，占 4 bits，用于存储行级别的状态信息，包括是否标记删除、是否是最小记录，另外两个 bit 是预留位；
- 对于非 INSTANT 算法，需要重建表，需要修改现有行的物理结构，因此每行数据都需要处理，执行效率低；
- 对于 INSTANT 算法，并不需要重建表，也不需要修改现有行的物理结构，只修改存储在系统表中的表结构，因此执行效率高。

如果不修改数据，仅修改表结构，将导致数据与表结构不一致，这也是 INSTANT 算法要解决的核心问题。

MySQL 在读取数据时，通过指针可以获取记录的开始位置，但是并没有保存记录的长度，也没有保存每个字段的长度，因此需要通过逐个读取字段获取完整的行记录，具体是**先获取每个字段的长度然后获取字段的值**。

其中：

- 对于定长字段，从表结构中获取字段长度；
- 对于变长字段，从行记录中获取变长字段长度列表。

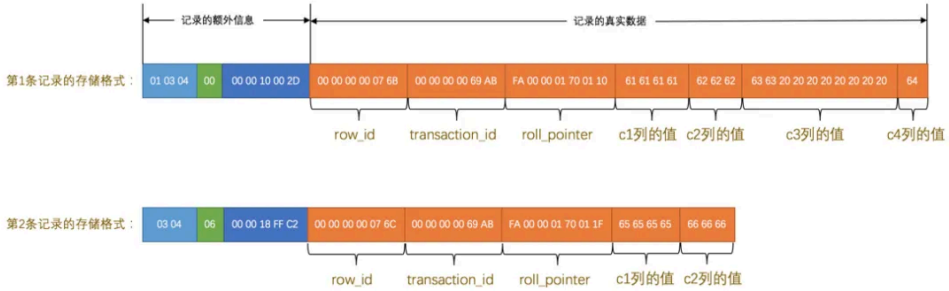
如下所示，表中有四个字段，其中第三个字段是定长字段，其他都是变长字段，使用 ascii 字符集，表中插入两行记录。

```
mysql> CREATE TABLE record_format_demo (
    ->     c1 VARCHAR(10),
    ->     c2 VARCHAR(10) NOTNULL,
    ->     c3 CHAR(10),
    ->     c4 VARCHAR(10)
    -> ) CHARSET=ascii ROW_FORMAT=COMPACT;
Query OK, 0 rows affected (0.03 sec)

mysql> INSERTINTO record_format_demo(c1, c2, c3, c4)
VALUES('aaaa', 'bbb', 'cc', 'd'),
('eeee', 'fff', NULL, NULL);
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM record_format_demo;
+------+-----+------+------+
| c1   | c2  | c3   | c4   |
+------+-----+------+------+
| aaaa | bbb | cc   | d    |
| eeee | fff | NULL | NULL |
+------+-----+------+------+
2 rows in set (0.00 sec)
```

下图是两条记录存储的行记录，接下来看怎么从中读取数据。



其中：

- 由于 InnoDB 表中没有指定主键，也没有唯一约束，自动为记录添加 row_id 隐藏字段；
- 第一条记录，01 03 04 是变成字段长度列表，0x61616161 表示字符串 'aaaa'，0x626262 表示字符串 'bbb'，0x64 表示字符串 'd'；
- 第一条记录，c3 列的类型是定长字段 char(10)，实际存储字符串 'cc'，而 ascii 字符集中的字节表示是 '0x6363'，虽然表示这个字符串只占用了 2 个字节，但整个 c3 列仍然占用了 10 个字节的空间，除真实数据以外的 8 个字节的统统都用空格字符填充，空格字符在 ascii 字符集的表示就是 0x20；
- 第二条记录，NULL 值列表 06，对应二进制 0000 0110，其中 1 表示列值为 NULL。c3 和 c4 列的值都为 NULL，它们被存储在了前面的 NULL 值列表处，在记录的真实数据处就不再冗余存储，从而节省存储空间。

DDL

对于 DDL，最关心的两点通常是锁定级别与执行时间，前者决定影响范围，后者决定影响时间，正常情况下这些都是 MySQL 内部自动处理的，但是执行前还是需要有所了解才可以，否则不一定可以满足业务需求，比如财务分库分表 DDL 加一个字段用时一周。

MySQL ALTER TABLE 语法中有以下两个子句，包括 ALGORITHM 与 LOCK。

ALGORITHM 用于指定 DDL 算法，支持以下选项：

- COPY，MySQL 5.5 及之前的方式，会重建表且表的记录格式会发生变化；
- INPLACE，MySQL 5.6 引入，支持在原表上进行操作，包括 rebuild 与 no-rebuild 两种方式，其中 rebuild 会重建表，no-rebuild 方式只会修改表的元数据信息；
- INSTANT，MySQL 8.0.12 引入，只修改表的元数据信息；
- DEFAULT，如果不指定 ALGORITHM，默认为 DEFAULT，将由 MySQL 自行选择 DDL 使用的算法，优先级由高到低为 INSTANT、INPLACE、COPY。

LOCK 用于指定 DDL 的锁定级别，支持以下选项：

- NONE，允许查询和 DML 操作。注意所有 DDL 操作都不阻塞读请求，因此这里所说的 DML 不包括 select；
- SHARED，允许查询，但是阻塞 DML 操作；
- EXCLUSIVE，阻塞查询和 DML 操作；
- DEFAULT，如果不指定 LOCK，默认为 DEFAULT，将由 MySQL 自行选择 LOCK 使用的级别，优先级由高到低为 NONE、SHARED、EXCLUSIVE。

因此：

- 当 ALGORITHM = COPY 时，LOCK 只能设置为 SHARED / EXCLUSIVE，也就是说都会阻塞 DML。如果不指定 LOCK 级别，默认使用 SHARED；
- 当 ALGORITHM = COPY / INPLACE 时，都需要拷贝数据重建表，因此表越大 DDL 执行时间越长。

简单对比以上三种算法，理论上性能依次提高。

ALGORITHM	Permits Concurrent DML	DESCRIPTION
COPY	No	copy original table data to new table
INPLACE	Yes	not copy data, but may rebuild the table in place
INSTANT	Yes	only modify metadata, table data is unaffected

其中：

- **online DDL 表示 DDL 执行过程中不阻塞 DML**；
- **只有当 ALGORITHM = INSTANT / INPLACE 且 LOCK = NONE 时，才可以称之为 online DDL**；
- LOCK 的锁定级别分多个，元数据锁也分多个级别，但是这里的 LOCK 和元数据锁无关。

下一个问题是执行后怎么判断使用了哪种 DDL 算法。

COPY 与 INPLACE 可以通过 DDL 执行完成后的返回值中受影响行数 rows affected 来判断，如果大于 0，表明发生数据拷贝，因此 ALGORITHM = COPY，否则 ALGORITHM = INPLACE。

> For DDL operations that modify table data, you can determine whether a DDL operation performs changes in place or performs a table copy by looking at the “rows affected” value displayed after the command finishes.

但是具体 INPLACE 如何判断是 rebuild 还是 no-rebuild 方式？目前未找到相关方法，可以简单地根据执行时间来判断。

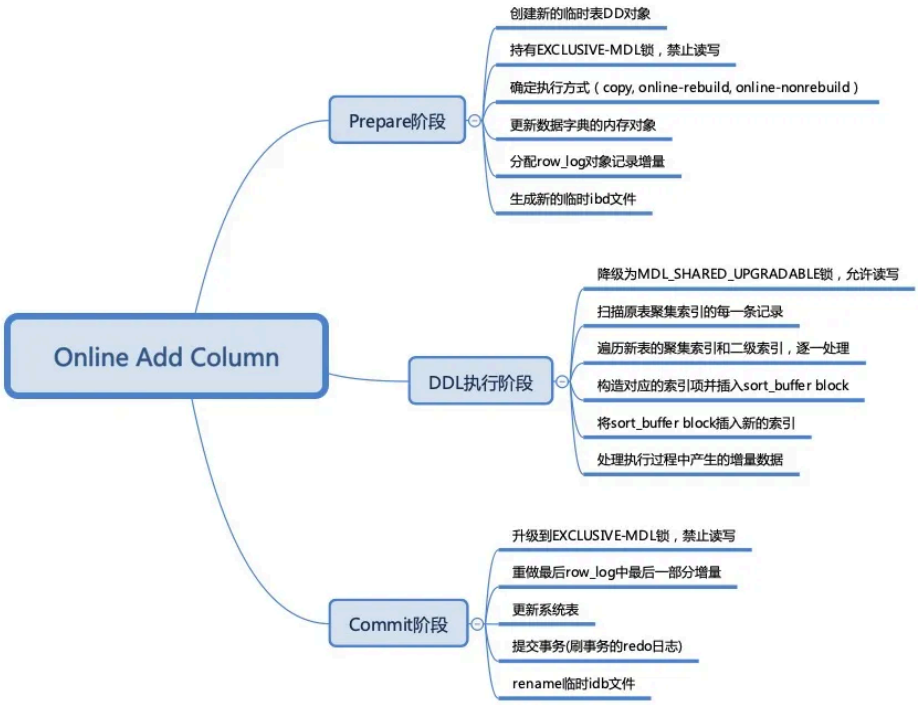
分析

online DDL

online DDL 的执行过程可以分为三个阶段，包括 prepare、run、commit。其中：

- prepare，创建临时表，更新数据字典；
- run，重建表空间，存量数据拷贝到临时表，对于大表，该阶段用时最长；
- commit，增量数据应用到临时表，更新系统表。

主要流程见下图。



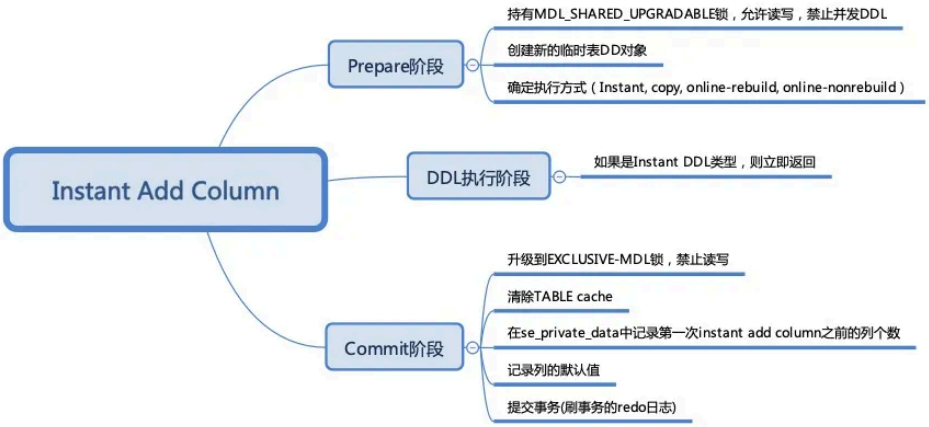
其中：

- prepare 与 commit 阶段分别需要持有 EXCLUSIVE 元数据锁禁止读写，因此如果有其他事务持有元数据锁，将导致元数据锁等待；
- **run 阶段降级为 SHARED 元数据锁允许读写，显然这就是所谓的 online，因此该阶段不会导致元数据锁等待，表明元数据锁等待只可能发生在 prepare 与 commit 阶段**；
- DDL 执行期间的增量数据保存在 row_log 对象中，在 commit 阶段回放这部分数据。

INSTANT

Instant add column 在增加列时，实际上只是修改了 schema，并没有修改原来存储在文件中的行记录，不需要执行最耗时的 rebuild 和 apply row log 过程，因此效率非常高。

主要流程：



原理

行格式 INSTANT_FLAG

8.0.12 版本开始支持 INSTANT 算法，仅修改元数据，不修改表数据。

> Operations that support ALGORITHM=INSTANT only modify metadata in the data dictionary. No exclusive metadata locks are taken on the table during preparation and execution phases of the operation, and table data is unaffected, making the operations instantaneous.

> If not specified explicitly, ALGORITHM=INSTANT is used by default by operations that support it. If ALGORITHM=INSTANT is specified but not supported, the operation fails immediately with an error.

需要注意的是仅支持在最后一列后加字段，不支持指在定位置加字段，这条限制的原因也将在下文中分析。

首先用一个示例分析 INSTANT 算法要解决的问题与具体实现。

下面有两个 DDL，三条数据，每条数据对应不同的表结构，对应的字段数量分别为 2、3、4。

```
CREATE TABLE t1 (a INTNOTNULL AUTO_INCREMENT PRIMARY KEY, b INT);
INSERTINTO t1 VALUES(0, 1); -- Row 1

ALTERTABLE t1 ADDCOLUMN c INTDEFAULT10;
INSERTINTO t1 VALUES(0, 2, 20); -- Row 2

ALTERTABLE t1 ADDCOLUMN d INT;
INSERTINTO t1 VALUES(0, 3, 20, 10); -- Row 3
```

写入的场景比较简单，直接使用最新的表结构即可，主要问题是查询的场景，**由于存量数据没有更新，因此会缺失部分字段，所以问题就是如何判断具体一行数据缺失哪些字段。**

可以通过数据行的版本号来判断数据行对应的表结构，但是为了降低实现的难度，并没有采用这种方案，而是通过打标判断是否执行过 instant add column 操作。

> To make an ADD COLUMN metadata change only, it is necessary to mark the record accordingly, so that later scan can know which record is fit to which table definition.

> One solution is to give every record or page a version number, meaning that the record or records in a page are defined by table structure version X, so to parse a record, a proper table structure can be picked accordingly. But this is more complex and lots of work need to be done.

> A simpler solution is how this worklog works. Instead of giving record/page a version, a bit in record would be set if there was any instant ADD COLUMN happened before. With this bit set, it's possible to distinguish a record is inserted before any instant ADD COLUMN or not.

> At the meantime, it's necessary to remember the existing column number when first instant ADD COLUMN happens.

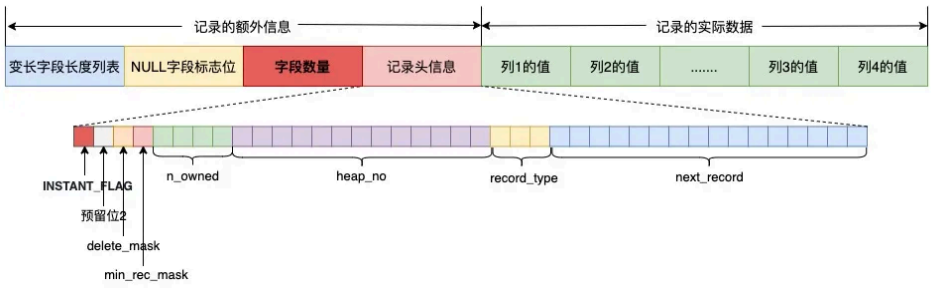
具体实现中引入了两个字段：

- 标志位，用于标识是否执行过 instant add column 操作，如果执行过，针对存量数据的缺失字段返回默认值；
- 字段数量，用于保存当前的字段数量，由于仅支持在末尾加字段，字段顺序不变，因此通过字段数量就可以确定字段列表，超出的字段同样返回默认值。

由于只有一个 bit 记录字段的变化，因此不支持改变字段顺序，所以仅支持在末尾加字段，并且不支持删除字段。

> Since there is only one bit to indicate the column change, so the order of columns are assumed to be not changed after instant ADD COLUMN. For short, the newly instantly added columns can only appended at last of the table.

如下图所示，红色部分分别表示 INSTANT_FLAG 标志位与字段数量。



其中：

- info_bits 的一个 bit 位 INSTANT_FLAG 表示是否执行过 instant add column 操作；
- 如果执行过 instant add column 操作，记录中新增一个字段保存字段数量。

MySQL · 引擎特性 · 8.0 Instant Add Column功能解析 中使用 hexdump 解析 ibd 文件显示 instant add column 操作后存量数据保持不变，增量数据中新增了两个字段，包括 INSTANT_FLAG 标志位与字段数量，且行记录中多了一个新增的字段。

查询时，对于具体的一行数据，首先判断 INSTANT_FLAG，如果执行过 instant add column 操作，接下来查询字段数量，超出的字段使用默认值。

元数据 VERSION

8.0.29 中增强 INSTANT 算法，支持在指定位置加字段，且支持在指定位置删除字段。

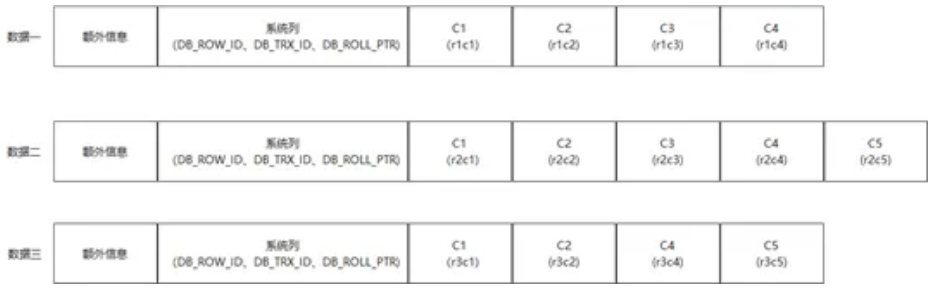
> Prior to MySQL 8.0.29, an instantly added column could only be added as the last column of the table. From MySQL 8.0.29, an instantly added column can be added to any position in the table.

> Instantly added or dropped columns create a new row version. Up to 64 row versions are permitted. A new TOTAL_ROW_VERSIONS column was added to the INFORMATION_SCHEMA.INNODB_TABLES table to track the number of row versions.

同样用一个示例分析 INSTANT 算法要解决的问题与具体实现，不同之处在于这个案例中包括删除列。

```
CREATE TABLE t1 (C1 char(10), C2 char(10), C3 char(10), C4 char(10));
INSERTINTO t1 values ("r1c1", "r1c2", "r1c3", "r1c4"); -- 数据一
ALTERTABLE t1 ADDCOLUMN C5 CHAR(10) DEFAULT"c5_def", ALGORITHM=INSTANT;
INSERTINTO t1 values ("r2c1", "r2c2", "r2c3", "r2c4", "r2c5"); -- 数据二
ALTERTABLE t1 DROPCOLUMN C3, ALGORITHM=INSTANT;
INSERTINTO t1 values ("r3c1", "r3c2", "r3c4", "r3c5"); -- 数据三
SELECT * FROM t1;
+------+------+------+--------+
| C1   | C2   | C4   | C5     |
+------+------+------+--------+
| r1c1 | r1c2 | r1c4 | c5_def |
| r2c1 | r2c2 | r2c4 | r2c5   |
| r3c1 | r3c2 | r3c4 | r3c5   |
+------+------+------+--------+
```

由于 INSTANT ADD / DROP 操作并不会修改存量数据，因此磁盘上的实际数据如下图所示，其中数据一多了 C3 列，缺少 C5 列，数据二多了 C3 列。

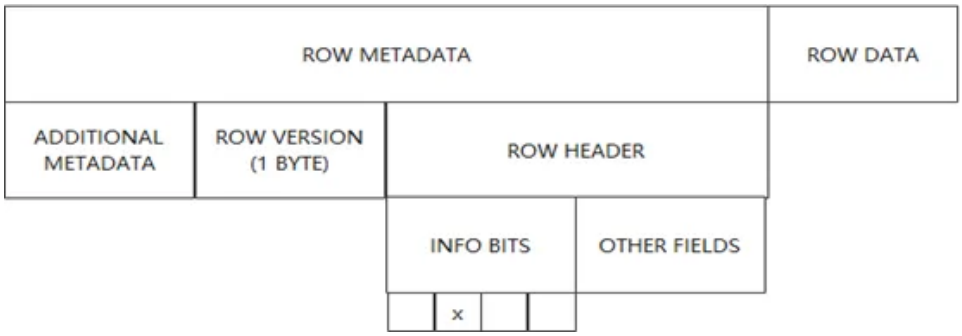


因此**问题就是如何正确解析并返回存量数据**。

具体实现中引入元数据 VERSION，包括表元数据、列元数据、行格式。

简单说就是每行记录中保存数据写入时的行版本（row version），INSTANT ADD / DROP 操作时更新行版本，元数据中维护每个行版本对于字段的可见性。

如下图所示，行格式中新增 ROW VERSION 字段保存数据写入时当前表的 VERSION 信息。



一文为你解读MySQL8.0 Instant DDL源码实现 文章中详细介绍了表元数据、列元数据与行格式的变化，下面仅展示 INSTANT DROP COLUMN 操作后的列元数据。

```
ALTER TABLE t1 DROP COLUMN C3, ALGORITHM=INSTANT;
INSERT INTO t1 values ("r3c1", "r3c2", "r3c4", "r3c5"); -- 数据三
SELECT NAME, HIDDEN, SE_PRIVATE_DATA FROM mysql.columns WHERE table_id = (SELECT ID FROM
```

列元数据查询结果见下图。

```
+-------------------------+---------+-----------------------------------------------------------------------+
| NAME                    | HIDDEN  | SE_PRIVATE_DATA                                                       |
+-------------------------+---------+-----------------------------------------------------------------------+
| !hidden!_dropped_v2_p5_C3 | SE      | physical_pos=5;version_dropped=2;                                     |
| C1                      | Visible | physical_pos=3;table_id=2909;                                         |
| C2                      | Visible | physical_pos=4;table_id=2909;                                         |
| C4                      | Visible | physical_pos=6;table_id=2909;                                         |
| C5                      | Visible | default=63355f64656620202020;physical_pos=7;table_id=2909;version_added=1; |
| DB_ROLL_PTR             | SE      | physical_pos=2;table_id=2909;                                         |
| DB_ROW_ID               | SE      | physical_pos=0;table_id=2909;                                         |
| DB_TRX_ID               | SE      | physical_pos=1;table_id=2909;                                         |
+-------------------------+---------+-----------------------------------------------------------------------+
```

其中：

- HIDDEN 列表示可见性。三个系统列的可见性为 SE，代表 INNODB 可见，SERVER 层不可见；

- 新增 C5 列后元数据 SE_PRIVATE_DATA 字段中保存列的默认值，表的 VERSION 值从 0 到 1，因此 version_added 记录为 1；
- 删除 C3 列后表的 VERSION 值从 1 变成 2，因此 version_dropped 记录为 2。但是删除 C3 列后元数据中保留 C3 列的原因是磁盘数据中依然有该列的数据，因此解析行记录时需要所有列的元数据，但是由于列逻辑删除，因此 HIDDEN = SE 表示不可见；
- 数据三 C4 列的 physical_pos 值为 6，但是因为数据三中没有 C3 列的数据，所以 C4 列数据实际在位置 5。表明 physical_pos 记录的是相对物理位置，要结合其他元信息才能确定该 record 中列的具体物理分布情况。

磁盘上数据分布见下图。

需要注意的是行版本使用的限制是最多允许 64 个版本，因此当 INSTANT ADD / DROP 操作次数超过 64 时，如果 DDL 中指定 ALGORITHM = INSTANT，执行报错，如果不指定，将退化为 ALGORITHM = INPLACE。 注意重建表后表的 VERSION 值清空，存量数据使用最新表结构。

> Instantly added or dropped columns create a new row version. Up to 64 row versions are permitted. A new TOTAL_ROW_VERSIONS column was added to the INFORMATION_SCHEMA.INNODB_TABLES table to track the number of row versions.

64次更改极限！MySQL DBA如何巧妙规避即时DDL操作的陷阱？ 文章中介绍了 INSTANT DDL 一个表支持64次即时更改的限制，并给出了相关建议。

结论

MySQL 中表结构与数据分开存储，表结构同时维护在磁盘的数据文件与内存的数据字典中。

通过指针可以获取每行数据的开始位置，但是数据中并不保存字段列表，也不保存每个字段的长度，因此读取数据时需要先获取每个字段的长度然后获取字段的值。对于定长字段从表结构中获取字段长度，对于变长字段从变成字段列表中获取字段长度。

显然读取数据时需要确认每行数据对应的字段信息，因此对于非 INSTANT DDL 算法，每次执行 DDL 时都会修改现有行的物理结构，优点是可以保证数据与表结构的一致性，缺点是 DDL 的执行时间与表大小成正比。

为解决 DDL 慢的问题，出现了 INSTANT DDL 算法，该算法不修改现有行的物理结构，仅修改表结构元数据，当然这样会导致数据与表结构不一致，这也是该算法要解决的核心问题，MySQL 通过两个版本的迭代解决了该问题。

- 8.0.12，仅支持在最后一列后加字段，不支持指在定位置加字段，其中：
  - 具体实现是在行格式中新增 INSTANT_FLAG 标志位与字段数量，在字段顺序不变的前提下通过字段数量就可以确定字段列表；
  - 使用限制是仅支持在末尾加字段，原因是要求字段顺序不变；
  - 查询时，对于具体的一行数据，首先判断 INSTANT_FLAG，如果执行过 instant add column 操作，接下来查询字段数量，超出的字段使用默认值。
- 8.0.29，支持在指定位置加字段，且支持在指定位置删除字段，其中：
  - 具体实现是引入元数据 VERSION，包括表元数据、列元数据、行格式；
  - 每行记录中保存数据写入时的行版本（row version），INSTANT ADD / DROP 操作时更新行版本，元数据中维护每个行版本对于字段的可见性；
  - 使用限制是行版本最多允许 64 个版本，因此当 INSTANT ADD / DROP 操作次数超过 64 时，将退化为 INPLACE；
  - 重建表后表的 VERSION 值清空，存量数据使用最新表结构。

待办

- DDL 原子性
- DDL 回滚
- PolarDB
- TiDB

参考教程

- 《MySQL 实战》
- 《MySQL 是怎样运行的：从根儿上理解 MySQL》
- MySQL · 源码阅读 · 白话Online DDL

http://mysql.taobao.org/monthly/2021/03/06/

- MySQL Online DDL的改进与应用

https://www.cnblogs.com/xinysu/p/6732646.html

- MySQL · 引擎特性 · 8.0 Instant Add Column功能解析

http://mysql.taobao.org/monthly/2020/03/01/

- 一文为你解读MySQL8.0 Instant DDL源码实现

https://www.cnblogs.com/huaweiyun/p/18464467

- WL#11250: Support Instant Add Column

https://dev.mysql.com/worklog/task/?id=11250

- MySQL 8.0: InnoDB now supports Instant ADD COLUMN

https://dev.mysql.com/blog-archive/mysql-8-0-innodb-now-supports-instant-add-column/

- MySQL 8.0 INSTANT ADD and DROP Column(s)

https://blogs.oracle.com/mysql/post/mysql-80-instant-add-drop-columns

- **3分钟掌握Mysql是如何存储一行记录**
- [ibd2sql] mysql frm 文件结构解析

https://cloud.tencent.com/developer/article/2409341

- InnoDB 表空间可视化工具innodb_ruby

https://cloud.tencent.com/developer/article/1817555