## 5.7.26 drop table

5.7.26整个DROP TABLE过程可以简单地概括为

1. 申请持有表的MDL写锁

-- 不能对该表执行DML操作和查询操作

1. 获取dict\_sys->mutex这个数据字典的全局锁

-- 函数：ha\_innobase::delete\_table->row\_drop\_table\_for\_mysql->row\_mysql\_lock\_data\_dictionary

-- 阻塞DML语句，不会阻塞查询语句

1. 启动一个InnoDB DDL事务

-- 函数：ha\_innobase::delete\_table->row\_drop\_table\_for\_mysql->trx\_start\_for\_ddl

1. 设置 table->to\_be\_dropped = true
2. row\_add\_table\_to\_background\_drop\_list

-- 这里还不理解

1. 从系统表中清理表信息

拼接了一个巨大的SQL，用来从系统表中清理信息，会释放索引树(主键索引树、二级索引树)，同时清理AHI。

从数据字典缓存中删除表

1. row\_drop\_single\_table\_tablespace 删除表空间

lazy drop逻辑，清理buffer pool的flush list，会多次持有和释放buffer pool mutex以及flush list mutex，释放cpu资源：

1. 写入MLOG\_FILE\_DELETE类型的redo日志
2. unlink ibd文件

-- 删除ibd文件。

-- ha\_innobase::delete\_table->row\_drop\_table\_for\_mysql->row\_drop\_single\_table\_tablespace->fil\_delete\_tablespace->os\_file\_delete

-- C语言unlink()函数：删除文件

1. 提交InnoDB DLL事务

-- ha\_innobase::delete\_table->row\_drop\_table\_for\_mysql->trx\_commit\_for\_mysql

1. 释放dict\_sys->mutex

-- ha\_innobase::delete\_table->row\_drop\_table\_for\_mysql->row\_mysql\_unlock\_data\_dictionary

1. 释放表的MDL写锁。

## Mysql8.0 drop table

drop table test1023;

b mysql\_execute\_command

b mysql\_rm\_table

res = mysql\_rm\_table(thd, first\_table, lex->drop\_if\_exists, lex->drop\_temporary); //删除表 first\_table表示删除的表， first\_table->table\_name表名

//1、检查：XA\_IDLE 或 XA\_PREPARED 事务状态中不允许使用 DROP 表。

//2、如果处于 MIXED 模式并删除 TEMP 表，则DROP表需要确定其日志记录格式。

//3、禁用删除已启用的日志表，必须在名称锁定之前完成

error = mysql\_rm\_table\_no\_locks(thd, tables, if\_exists, drop\_temporary, false, &not\_used, &post\_ddl\_htons,&fk\_invalidator, &safe\_to\_release\_mdl); 执行删除普通表或临时表。

drop\_base\_table(thd, drop\_ctx, table, true /\* atomic \*/, post\_ddl\_htons, fk\_invalidator, &safe\_to\_release\_mdl\_atomic)) 删除单个基表

tdc\_remove\_table(thd, TDC\_RT\_REMOVE\_ALL, table->db, table->table\_name, false); 从表定义缓存中删除 TABLE 和 TABLE\_SHARE 的全部或部分（取决于参数）实例。

table\_cache\_manager.lock\_all\_and\_tdc(); 获取表缓存和表定义缓存（即 LOCK\_open）的所有实例的锁。

for (uint i = 0; i < table\_cache\_instances; i++) m\_table\_cache[i].lock() //给每个表缓存实例加锁；table\_cache\_instances

mysql\_mutex\_lock(&LOCK\_open) //表定义缓存加锁；table\_definition\_cache

key\_length = create\_table\_def\_key(db, table\_name, key) 为表创建表缓存/表定义缓存键。？？

table\_cache\_manager.free\_table(thd, remove\_type, share)

remove\_table(it)

table\_cache\_manager.unlock\_all\_and\_tdc() 释放对表缓存和表定义缓存的所有实例的锁。

mysql\_mutex\_unlock(&LOCK\_open);

for (uint i = 0; i < table\_cache\_instances; i++) m\_table\_cache[i].unlock()

int error = ha\_delete\_table(thd, hton, path, table->db, table->table\_name, table\_def, !drop\_ctx.drop\_database); 从存储引擎中删除表。

error = file->ha\_delete\_table(path, table\_def) => handler::ha\_delete\_table(const char \*name, const dd::Table \*table\_def) delete表的公用接口

mark\_trx\_read\_write() 标记为读写事物

delete\_table(name, table\_def)

innobase\_register\_trx(ht, thd, trx); 调用两阶段提交进行注册事物

trans\_register\_ha(thd, FALSE, hton, &trx\_id)在引擎中注册一个事物

trans\_register\_ha(thd, TRUE, hton, &trx\_id)

trx\_register\_for\_2pc(trx);

innobase\_basic\_ddl::delete\_impl(thd, name, table\_def) 删除表

error = row\_drop\_table\_for\_mysql(norm\_name, trx, true, handler) drop table for mysql

row\_mysql\_lock\_data\_dictionary(trx) 数据字典加锁

mutex\_enter(&dict\_sys->mutex)

table = dict\_table\_check\_if\_in\_cache\_low(name) 检查表是否在数据字典缓存中

dict\_stats\_wait\_bg\_to\_stop\_using\_table(table, trx); 等到后台统计线程停止使用指定的表。

dict\_stats\_recalc\_pool\_del(table)从自动重新计算池（auto recalc pool）中删除给定的表。 ？？不太明白

err = dict\_stats\_drop\_table(name, errstr, sizeof(errstr)) 如果该表存在并且其中有该表的统计信息，则从持久存储中删除该表及其所有索引的统计信息。

dict\_table\_prevent\_eviction(table); 通过将表从 LRU 列表移动到非 LRU 列表（如果它尚未存在）来防止表驱逐。

lock\_remove\_all\_on\_table(table, TRUE) 删除要删除的表上的锁。

index->page = FIL\_NULL 在开始删除表之前，标记数据字典缓存中所有不可用的索引。

err = row\_drop\_table\_from\_cache(table, trx); 作为删除表的一部分，从内存缓存中删除表。

btr\_drop\_ahi\_for\_table(table); 删除表的任何自适应哈希索引条目。

dict\_table\_remove\_from\_cache(table)从字典缓存中删除一个表对象。

log\_ddl->write\_drop\_log(trx, table\_id);

err = insert\_drop\_log(trx, id, thread\_id, table\_id)

err = log\_ddl->write\_delete\_space\_log(trx, nullptr, space\_id, filepath, true, true);

row\_mysql\_unlock\_data\_dictionary(trx); 解锁数据字典排他锁。

dd\_drop\_tablespace(client, thd, dd\_space\_id) 删除表空间

destroy(file) file是handler类型

bool result = dd::drop\_table(thd, table->db, table->table\_name, \*table\_def)

hton->post\_ddl(thd)

static void innobase\_post\_ddl(THD \*thd) 在 DDL 语句之后执行提交后/回滚清理

dberr\_t err = log\_ddl->post\_ddl(thd) => Log\_DDL::post\_ddl(THD \*thd)

dberr\_t err = replay\_by\_thread\_id(thread\_id) =>Log\_DDL::replay\_by\_thread\_id(ulint thread\_id)

log\_ddl->replay(\*record) => Log\_DDL::replay(DDL\_Record &record)

replay\_delete\_space\_log(record.get\_space\_id(),record.get\_old\_file\_path()) => Log\_DDL::replay\_delete\_space\_log(space\_id\_t space\_id,const char \*file\_path)

dict\_sdi\_remove\_from\_cache(space\_id, nullptr, true); 从表缓存中删除 SDI 表。

row\_drop\_tablespace(space\_id, file\_path) 删除表空间作为删除或重命名表的一部分。这将删除 fil\_space\_t（如果找到）和磁盘上的文件。

err = fil\_delete\_tablespace(space\_id, BUF\_REMOVE\_FLUSH\_NO\_WRITE)

shard->space\_delete(space\_id, buf\_remove) => Fil\_shard::space\_delete(space\_id\_t space\_id, buf\_remove\_t buf\_remove)

buf\_LRU\_flush\_or\_remove\_pages(space\_id, buf\_remove, nullptr);

buf\_LRU\_remove\_pages(buf\_pool, id, buf\_remove, trx, strict)

buf\_flush\_dirty\_pages(buf\_pool, id, observer, false, nullptr, strict);

mutex\_enter(&buf\_pool->LRU\_list\_mutex);

err = buf\_flush\_or\_remove\_pages(buf\_pool, id, observer, flush, trx); 从flush链表中进行删除

mutex\_exit(&buf\_pool->LRU\_list\_mutex);

os\_file\_delete(innodb\_data\_file\_key, path) 删除文件

总结：

1. 表缓存和表定义缓存（即 LOCK\_open）的所有实例的加锁

从表缓存中删除所有此表的缓存；

1. 数据字典加锁：
2. 检查表是否在数据字典缓存中，并且等到后台统计线程停止使用指定的表。
3. 从自动重新计算池（auto recalc pool）中删除给定的表。 ？？不太明白 应该是统计信息的缓存
4. 如果该表存在并且其中有该表的统计信息，则从持久存储中删除该表及其所有索引的统计信息。
5. 通过将表从 LRU 列表移动到非 LRU 列表（如果它尚未存在）来防止表驱逐。
6. 删除该表上的锁。
7. 在开始删除表之前，标记数据字典缓存中所有不可用的索引。
8. 删除表的任何自适应哈希索引条目。
9. 从字典缓存中删除一个表对象。
10. 写ddl drop log日志
11. Flush 链表加锁

删除buffer pool中的脏页