# 背景

# 概述

Attachable transaction是从5.7引入的一个概念，主要用来对事务类型的系统表访问的接口，从事务的系统表查询得到一致的数据。

目前主要是对innodb类型的系统表访问的接口，也只有innodb引擎实现了attachable transaction的支持。Attachable transaction 主要是为访问事务类型的系统表而设计的，它是一个嵌入用户事务的内部事务，当用户事务用到元信息时就需要开启一个attachable transaction去访问数据字典，得到用户表的元信息后，要结束attachable transaction，用户会话要能恢复到用户事务之前的状态。

Attachable transaction是一个AC-RO-RC-NL (auto-commit, read-only, read-committed, non-locking) 事务。引入Attachable transaction主要有以下几个原因：

1、如果用户开启的一个事务需要访问系统表获取表的元信息，而访问系统表可能和用户事务指定的隔离级别不一致，这时就需要开启一个独立的访问数据字典的事务，要求访问数据字典事务的隔离级别必须是READ COMMITTED，其隔离级别可能和用户指定的隔离级别不一致。

2、对数据字典的访问必须是非锁定的。

3、即时用户事务已经打开和锁定了用户表，在执行SQL语句的在任何时候也应该能对数据字典打开，来查询用户表的各种元信息。

# 代码

参考：<http://mysql.taobao.org/monthly/2020/06/03/>

## 数据结构

在每个会话的THD结构里，添加了一个Attachable\_trx \*m\_attachable\_trx;字段，用来指向当前会话的内嵌事务。

### Attachable\_trx

Attachable\_trx类型定义如下：

/\*\*

Class representing read-only attachable transaction, encapsulates

knowledge how to backup state of current transaction, start

read-only attachable transaction in SE, finalize it and then restore

state of original transaction back. Also serves as a base class for

read-write attachable transaction implementation.

\*/

class Attachable\_trx

{

public:

Attachable\_trx(THD \*thd);

virtual ~Attachable\_trx();

virtual bool is\_read\_only() const { return true; }

protected:

/// THD instance.

THD \*m\_thd;

/// Transaction state data.

Transaction\_state m\_trx\_state;

private:

Attachable\_trx(const Attachable\_trx &);

Attachable\_trx &operator =(const Attachable\_trx &);

};

### Transaction\_state

其中最重要的是m\_trx\_state字段，期存放着attachable transaction的重要信息，就是用它来保存外部用户事务的状态，以便在着attachable transaction结束后能恢复到原来的用户事务状态。

其定义如下：

/\*\* An utility struct for @c Attachable\_trx \*/

struct Transaction\_state

{

void backup(THD \*thd);

void restore(THD \*thd);

/// SQL-command.

enum\_sql\_command m\_sql\_command;

Query\_tables\_list m\_query\_tables\_list;

/// Open-tables state.

Open\_tables\_backup m\_open\_tables\_state;

/// SQL\_MODE.

sql\_mode\_t m\_sql\_mode;

/// Transaction isolation level.

enum\_tx\_isolation m\_tx\_isolation;

/// Ha\_data array.

Ha\_data m\_ha\_data[MAX\_HA];

/// Transaction\_ctx instance.

Transaction\_ctx \*m\_trx;

/// Transaction read-only state.

my\_bool m\_tx\_read\_only;

/// THD options.

ulonglong m\_thd\_option\_bits;

/// Current transaction instrumentation.

PSI\_transaction\_locker \*m\_transaction\_psi;

/// Server status flags.

uint m\_server\_status;

};

## 核心接口API

### 启动attachable transaction

主要有这几个函数：

THD::begin\_attachable\_transaction()

begin\_attachable\_ro\_transaction()

begin\_attachable\_rw\_transaction(),

启动attachable transaction的过程中主要完成以下功能：

1、在开始一个attachable\_transaction之前，先要保存当前已经开始的用户的正常事务状态。

2、开始设置一个新的事务所需要的各种状态。

3、重新设置THD::ha\_data。通过重制THD::ha\_data值使InnoDB在接下来的操作去创建以下新的事务。

4、执行对系统表的操作。

5、当执行到存储引擎层时，InnoDB从传进来的THD指针中发现事务还没开启 （因为THD::ha\_data重置了），InnoDB就会新启一个事务。

6、InnoDB通过调用trans\_register\_ha()通知server它已经创建了一个新事务。

7、InnoDB执行请求的操作，返回给server层。

THD::Attachable\_trx::Attachable\_trx(THD \*thd)

:m\_thd(thd)

{

m\_trx\_state.backup(m\_thd); //保存当前已经开始的用户的正常事务状态

......

m\_thd->reset\_n\_backup\_open\_tables\_state(&m\_trx\_state.m\_open\_tables\_state); //保存一些打开表的状态信息，并且重新为新的事物重置表状态

// 为attachable transaction创建一个新的事物上下文

m\_thd->m\_transaction.release(); // it's been backed up.

m\_thd->m\_transaction.reset(new Transaction\_ctx());

......

for (int i= 0; i < MAX\_HA; ++i)

m\_thd->ha\_data[i]= Ha\_data(); //重新设置THD::ha\_data

m\_thd->tx\_isolation= ISO\_READ\_COMMITTED; // attachable transaction 必须是read committed

m\_thd->tx\_read\_only= true; ／／ attachable transaction 必须是只读的

// attachable transaction 必须是 AUTOCOMMIT

m\_thd->variables.option\_bits|= OPTION\_AUTOCOMMIT;

m\_thd->variables.option\_bits&= ~OPTION\_NOT\_AUTOCOMMIT;

m\_thd->variables.option\_bits&= ~OPTION\_BEGIN;

......

}

### 结束attachable transaction

结束attachable transaction 函数：THD::end\_attachable\_transaction()。

因为attachable transaction事务是一个只读的自提交事务，所以它不需要调用任何事务需要提交回滚的函数，比如：

ha\_commit\_trans()

ha\_rollback\_trans()

trans\_commit()

trans\_rollback()。

所以定义了此函数用来结束当前的attachable transaction。它主要完成以下功能：

1、调用close\_thread\_tables()关闭在attachable transaction中打开的表。

2、调用close\_connection()通知引擎层去销毁为attachable transaction创建的事务。

3、InnoDB调用trx\_commit\_in\_memory()去销毁readview等操作。

4、最后要恢复之前正常的用户事务,包括THD::ha\_data的恢复，这个通过调用下面提到的事务状态的backup()/restore()接口。

THD::Attachable\_trx::~Attachable\_trx()

{

......

close\_thread\_tables(m\_thd); //调用close\_thread\_tables()关闭在attachable transaction中打开的

ha\_close\_connection(m\_thd); //调用close\_connection()通知引擎层去销毁为attachable transaction创建的事务

// 恢复之前正常的用户事状态

m\_trx\_state.restore(m\_thd);

m\_thd->restore\_backup\_open\_tables\_state(&m\_trx\_state.m\_open\_tables\_state);

......

}

### 保存/恢复attachable transaction

attachable transaction事务的保存和恢复接口函数Transaction\_state::backup()／restore()。

由于一个会话同时只允许有一个活跃事务，当需要访问内部的事务系统表时，就需要开启一个Attachable transaction事务，这时就要先把外部的主事务状态先保存起来，等内部开启的Attachable transaction事务执行完，再把外部用户执行的事务恢复回来。为了实现这个功能，提供了backup和restore的接口。

### Handler API

为了让存储层知道server层开启的是一个attachable transaction，handler API新加了一个HA\_ATTACHABLE\_TRX\_COMPATIBLE 标志。设置这个标志的存储引擎类型表示引擎层已经意识到开启了attachable transaction的事务类型。目前InnoDB和MyISAM引擎都能处理这种attachable transaction。但处理方式不同：

1、对InnoDB而言，完全支持attachable transaction事务，能够感知到THD::ha\_data 的变化并开启一个attachable transaction事务。在close\_connection时结束一个attachable transaction事务，然后恢复用户正常的事务继续处理。 2、对MyISAM而言，虽然知道server开启了一个attachable transaction事务但也不做任何处理，就是简单但忽律掉 THD::ha\_data and close\_connection handlerton相关但处理。

在初始化一个InnoDB表时，设置HA\_ATTACHABLE\_TRX\_COMPATIBLE 标志的代码如下:

ha\_innobase::ha\_innobase(

/\*=====================\*/

handlerton\* hton,

TABLE\_SHARE\* table\_arg)

:handler(hton, table\_arg),

m\_prebuilt(),

m\_prebuilt\_ptr(&m\_prebuilt),

m\_user\_thd(),

m\_int\_table\_flags(HA\_REC\_NOT\_IN\_SEQ

......

| HA\_ATTACHABLE\_TRX\_COMPATIBLE

| HA\_CAN\_INDEX\_VIRTUAL\_GENERATED\_COLUMN

),

m\_start\_of\_scan(),

m\_num\_write\_row(),

m\_mysql\_has\_locked()

{}