Atitit.自定义存储引擎的接口设计 api 标准化 attilax 总结 mysql

[1. 图16.1：MySQL体系结构 1](#_Toc10077)

[2. 16.7. 创建表create()虚拟函数： 2](#_Toc7325)

[3. 16.8. 打开表 open（） 2](#_Toc24744)

[4. --------------------------------------------------------------------------------------------------------------------- 2](#_Toc5399)

[5. 16.9. 实施基本的表扫描功能 2](#_Toc29254)

[5.1. 目录 3](#_Toc14003)

[5.1.1. 16.9.1. 实施store\_lock()函数 3](#_Toc13097)

[5.1.2. 16.9.2. 实施external\_lock()函数 3](#_Toc1920)

[5.1.3. 16.9.3. 实施rnd\_init()函数 3](#_Toc30849)

[5.1.4. 16.9.4. 实施info()函数 3](#_Toc14816)

[5.1.5. 16.9.5. 实施extra()函数 3](#_Toc23888)

[5.1.6. 16.9.6. 实施rnd\_next()函数 3](#_Toc3661)

[5.2. CSV引擎的9行表扫描过程中进行的方法调用： 3](#_Toc29599)

[5.3. 16.9.1. 实施store\_lock()函数 4](#_Toc15418)

[5.4. 实施rnd\_init()函数 5](#_Toc5216)

[5.5. 16.9.4. 实施info()函数 5](#_Toc25338)

[5.6. 16.9.5. 实施extra()函数 5](#_Toc27448)

[5.7. 16.9.6. 实施rnd\_next()函数 6](#_Toc19150)

[6. ------------------------------------------------------------------------------------------------------------------- 6](#_Toc1480)

[7. 关闭表close(void) 6](#_Toc16275)

[8. 6](#_Toc10810)

[9. 16.11. 为存储引擎添加对INSERT的支持write\_row() 6](#_Toc22944)

[10. 16.12. 为存储引擎添加对UPDATE的支持update\_row() 7](#_Toc5902)

[11. 16.13. 为存储引擎添加对DELETE的支持delete\_row() 7](#_Toc28372)

[12. 16.14. API引用 与详细说明 7](#_Toc7808)

[13. 8](#_Toc29933)

[14. 参考 8](#_Toc25094)

# 图16.1：MySQL体系结构

存储引擎负责管理数据存储，以及MySQL的索引管理。通过定义的API，MySQL服务器能够与存储引擎进行通信。

每个存储引擎均是1个继承类，每个类实例作为处理程序而被引用。

# 16.7. 创建表create()虚拟函数：

一旦实例化了处理程序，所需的第1个操作很可能是创建表。

你的存储引擎必须实现create()虚拟函数：

virtual int create(const char \*name, TABLE \*form, HA\_CREATE\_INFO \*info)=0;该函数应创建所有必须的文件，然后关闭表。MySQL服务器将调用随后需打开的表。

作者:: 老哇的爪子 Attilax 艾龙， EMAIL:1466519819@qq.com

转载请注明来源： <http://blog.csdn.net/attilax>

# 16.8. 打开表 open（）

在表上执行任何读或写操作之前，MySQL服务器将调用open()方法打开表数据和索引文件（如果存在的话）。

int open(const char \*name, int mode, int test\_if\_locked);第1个参数是要打开的表的名称。第2个参数确定了要打开的文件或准备执行的操作。它们的值定义于handler.h中，

# ---------------------------------------------------------------------------------------------------------------------

# 16.9. 实施基本的表扫描功能

## 目录

### 16.9.1. 实施store\_lock()函数

### 16.9.2. 实施external\_lock()函数

### 16.9.3. 实施rnd\_init()函数

### 16.9.4. 实施info()函数

### 16.9.5. 实施extra()函数

### 16.9.6. 实施rnd\_next()函数

最基本的存储引擎能实现只读表扫描功能。这类引擎可用于支持SQL日志查询、以及在MySQL

之外填充的其他数据文件。

## CSV引擎的9行表扫描过程中进行的方法调用：

ha\_tina::store\_lock

ha\_tina::external\_lock

ha\_tina::info

ha\_tina::rnd\_init

ha\_tina::extra - ENUM HA\_EXTRA\_CACHE Cache record in HA\_rrnd()

ha\_tina::rnd\_next

ha\_tina::rnd\_next

ha\_tina::rnd\_next

ha\_tina::rnd\_next

ha\_tina::rnd\_next

ha\_tina::rnd\_next

ha\_tina::rnd\_next

ha\_tina::rnd\_next

ha\_tina::rnd\_next

ha\_tina::extra - ENUM HA\_EXTRA\_NO\_CACHE End cacheing of records (def)

ha\_tina::external\_lock

ha\_tina::extra - ENUM HA\_EXTRA\_RESET Reset database to after open

## 16.9.1. 实施store\_lock()函数

在执行任何读取或写操作之前，调用store\_lock()函数。

将锁定添加到表锁定处理程序之前（请参见thr\_lock.c），mysqld将用请求的锁调用存储锁定。目前，存储锁定能将写锁定更改为读锁定（或其他锁定），忽略锁定（如果不打算使用MySQL锁定的话），或为很多表添加锁定（就像使用MERGE处理程序时作的那样）。

例如，Berkeley

DB能将所有的WRITE锁定更改为TL\_WRITE\_ALLOW\_WRITE（表示我们正在执行WRITES，但我们仍允许其他人员进行操作）。

释放锁定时，也将调用store\_lock()，在这种情况下，通常不需做任何事。

在某些特殊情况下，MySQL可能会发送对TL\_IGNORE的请求。这意味着我们正在请求与上次相同的锁定，这也应被忽略（当我们打开了表的某一部分时，如果其他人执行了表刷新操作，就会出现该情况，此时，mysqld将关闭并再次打开表，然后获取与上次相同的锁定）。我们打算在将来删除该特性。

在任何表扫描之前调用的函数是rnd\_init()函数。函数rnd\_init()用于为表扫描作准备，将计数器和指针复位为表的开始状态。

下述示例来自CSV存储引擎：

int ha\_tina::rnd\_init(bool scan)

{

DBUG\_ENTER("ha\_tina::rnd\_init");

current\_position= next\_position= 0;

records= 0;

chain\_ptr= chain;

DBUG\_RETURN(0);

}

## 实施rnd\_init()函数

在任何表扫描之前调用的函数是rnd\_init()函数。函数rnd\_init()用于为表扫描作准备，将计数器和指针复位为表的开始状态。

下述示例来自CSV存储引擎：

## 16.9.4. 实施info()函数

执行表扫描操作之前，将调用info()函数，以便为优化程序提供额外信息。

优化程序所需的信息不是通过返回值给定的，你需填充存储引擎类的特定属性，当info()调用返回后，优化程序将读取存储引擎类。

除了供优化程序使用外，在调用info()函数期间，很多值集合还将用于SHOW TABLE STATUS语句。

在sql/handler.h中列出了完整的公共属性，下面给出了一些常见的属性：

ulonglong data\_file\_length; /\* Length off data file \*/ulonglong max\_data\_file\_length; /\* Length off data file \*/ulonglong index\_file\_length;ulonglong max\_index\_file\_length;ulonglong delete\_length; /\* Free bytes \*/ulonglong auto\_increment\_value;ha\_rows records; /\* Records in table \*/ha\_rows deleted; /\* Deleted records \*/ulong raid\_chunksize;ulong mean\_rec\_length; /\* physical reclength \*/time\_t create\_time; /\* When table was created \*/time\_t check\_time;time\_t update\_time; 对于表扫描，最重要的属性是“records”，它指明了表中的记录数。当存储引擎指明表中有0或1行时，或有2行以上时，在这两种情况下，优化程序的执行方式不同。因此，当你在执行表扫描之前不清楚表中有多少行时，应返回大于等于2的值，这很重要（例如，数据是在外部填充的）。

## 16.9.5. 实施extra()函数

执行某些操作之前，应调用extra()函数，以便为存储引擎就如何执行特定操作予以提示。

额外调用中的提示实施不是强制性的，大多数存储引擎均返回0：

int ha\_tina::extra(enum ha\_extra\_function operation)

{

DBUG\_ENTER("ha\_tina::extra");

DBUG\_RETURN(0);

}

## 16.9.6. 实施rnd\_next()函数

完成表的初始化操作后，MySQL服务器将调用处理程序的rnd\_next()函数，每两个扫描行调用1次，直至满足了服务器的搜索条件或到达文件结尾为止，在后一种情况下，处理程序将返回HA\_ERR\_END\_OF\_FILE。

# -------------------------------------------------------------------------------------------------------------------

# 关闭表close(void)

当MySQL服务器完成表操作时，它将调用close()方法关闭文件指针并释放任何其他资源。

对于使用共享访问方法的存储引擎（如CSV引擎和其他示例引擎中显示的方法），必须将它们自己从共享结构中删除：

int ha\_tina::close(void) { DBUG\_ENTER("ha\_tina::close"); DBUG\_RETURN(free\_share(share)); } 对于使用其自己共享管理系统的存储引擎，应使用任何所需的方法，在它们的处理程序中，从已打开表的共享区删除处理程序实例。

# 

# 16.11. 为存储引擎添加对INSERT的支持write\_row()

一旦在你的存储引擎中有了读支持，下一个需要实施的特性是对INSERT语句的支持。有了INSERT支持，存储引擎就能处理WORM（写一次，读多次）应用程序，如用于以后分析的日志和归档应用等。

所有的INSERT操作均是通过write\_row()函数予以处理的：

int ha\_foo::write\_row(byte \*buf) \*buf参数包含将要插入的行，采用内部MySQL格式。基本的存储引擎将简单地前进到数据文件末尾，并直接在末尾处添加缓冲的内容，这样就能使行读取变得简单，这是因为，你可以读取行并将其直接传递到rnd\_next()函数的缓冲参数中。

# 16.12. 为存储引擎添加对UPDATE的支持update\_row()

通过执行表扫描操作，在找到与UPDATE语句的WHERE子句匹配的行后，MySQL服务器将执行UPDATE语句，然后调用update\_row()函数：

int ha\_foo::update\_row(const byte \*old\_data, byte \*new\_data)\*old\_data参数包含更新前位于行中的数据，而\*new\_data参数包含行的新内容（采用MySQL内部行格式）。

更新的执行取决于行格式和存储实施方式。某些存储引擎将替换恰当位置的数据，而其他实施方案则会删除已有的行，并在数据文件末尾添加新行。

非事务性存储引擎通常会忽略\*old\_data参数的内容，仅处理\*new\_data缓冲。事务性存储引擎可能需要比较缓冲，以确定在上次回滚中出现了什么变化。

如果正在更新的表中包含时间戳列，对时间戳的更新将由update\_row()调用管理。下述示例来自CSV引擎：

# 16.13. 为存储引擎添加对DELETE的支持delete\_row()

MySQL服务器采用了与INSERT语句相同的方法来执行DELETE语句：服务器使用rnd\_next()函数跳到要删除的行，然后调用delete\_row()函数删除行。

# 16.14. API引用 与详细说明

16.14.1. bas\_ext

16.14.2. close

16.14.3. create

16.14.4. delete\_row

16.14.5. delete\_table

16.14.6. external\_lock

16.14.7. extra

16.14.8. info

16.14.9. open

16.14.10. rnd\_init

16.14.11. rnd\_next

16.14.12. store\_lock

16.14.13. update\_row

16.14.14. write\_row

16.14.1. bas\_ext

# 

# 参考

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。

原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

MySQL 51简体中文手册 第16章：编写自定义存储引擎 南京廖华.htm