[MYSQL] 忘记root密码时, 不需要重启也能强制修改了!

原创 大大刺猬 2025-02-06 637

导读

之前讲过mysql忘记密码时的一些处理方法, 前面几种都是需要重启才生效的(包括修改ibd文件), 而不需要重启的方法(修改内存,或者gdb跳过认证)并没有给出完整实现. 而有的同学恰好就需要一个不用重启也能强制修改密码的方法...

所以今天来讲讲其中的 修改内存 实现强制修改密码操作.

原理分析

linux

原理很简单, 既然验证的密码是在内存中的, 那我们找到该密码直接修改为我们需要的密码即可.

~~以上, 就是这么滴简单~~

其实难点在于怎么访问mysqld进程的内存. `/proc/PID/` 下面有很多信息, 比如 io 表示这个进程读写了多少数据, 我们导入进度脚本就是查看的这个文件的rchar. 除此之外还有常用的stat,comm,fd等. 详情可以查看内核官网.

File	Content
clear_refs	Clears page referenced bits shown in smaps output
cmdline	Command line arguments
cpu	Current and last cpu in which it was executed (2.4)(smp)
cwd	Link to the current working directory
environ	Values of environment variables
exe	Link to the executable of this process
fd	Directory, which contains all file descriptors
maps	Memory maps to executables and library files (2.4)
mem	Memory held by this process
root	Link to the root directory of this process
stat	Process status
statm	Process memory status information
status	Process status in human readable form
wchan	Present with CONFIG_KALLSYMS=y: it shows the kernel function symbol the task is blocked in - or "0" if not blocked.
pagemap	Page table
stack	Report full stack trace, enable via CONFIG_STACKTRACE
smaps	An extension based on maps, showing the memory consumption of each mapping and flags associated with it
smaps_rollup	Accumulated smaps stats for all mappings of the process. This can be derived from smaps, but is faster and more convenient
numa_maps	An extension based on maps, showing the memory locality and binding policy as well as mem usage (in pages) of each mapping.

其中有个 `maps` 和 `mem` 文件, 就是该进程所使用的内存映射和具体的内存fd. linux上一切皆文件, 硬件也是

大大刺猬
关注
146 文章 108 粉丝 66K+ 浏览量
获得了 350 次点赞
内容获得 70 次评论
获得了 257 次收藏

TA的专栏
PYTHON解析MYSQL
收录 59 篇内容

热门文章
MYSQL 文件解析 (3) 数据文件(ibd)解析 2023-04-23 2801浏览
ibd2sql解析ibd文件为SQL 2023-04-27 2060浏览
[MYSQL] 数据恢复, 无备份, 只剩一个 ibd 文件 怎么恢复数据? 2024-04-12 1949浏览
MYSQL 文件解析 (1) binlog 文件解析 2023-04-23 1708浏览
mysql-5.7.38启动流程源码解读 2022-09-26 1529浏览

在线实训环境入口
MySQL在线实训环境 查看详情 »

最新文章
[MYSQL] 修改字段长度的时候不能使用instant算法? 其实inplace就够了 2025-07-30 49浏览
[MYSQL] 修改字段长度的时候不能使用instant算法? 那就定制一个? 2025-07-25 81浏览
[MYSQL] row_format=compressed的存储结构浅析 2025-07-18 55浏览
[MYSQL] 备份失败,但是啥日志信息都没有 2025-07-15 72浏览
[MYSQL] 从库 io_thread 接受binlog速度太慢? 2025-07-11 327浏览

文件, 包括内存,磁盘等.

我们查看mysqld进程的maps文件,得到如下信息

```
09:47:15 [root@ddcw21 ei]#cat /proc/`pidof mysqld`/maps
00400000-00c6f000 r--p 00000000 fd:00 307653646                          /soft/mysql_3386/mysqlbase/mysql/bin/mysqld
00c6f000-02a10000 r-xp 0086f000 fd:00 307653646                          /soft/mysql_3386/mysqlbase/mysql/bin/mysqld
02a10000-03b66000 r--p 02610000 fd:00 307653646                          /soft/mysql_3386/mysqlbase/mysql/bin/mysqld
03b67000-03cef000 r--p 03766000 fd:00 307653646                          /soft/mysql_3386/mysqlbase/mysql/bin/mysqld
03cef000-04091000 rw-p 038ee000 fd:00 307653646                          /soft/mysql_3386/mysqlbase/mysql/bin/mysqld
04091000-045c9000 rw-p 00000000 00:00 0
0494c000-06f7f000 rw-p 00000000 00:00 0                                  [heap]
7f8180000000-7f818002b000 rw-p 00000000 00:00 0
7f818002b000-7f8184000000 ---p 00000000 00:00 0
7f8188000000-7f8188021000 rw-p 00000000 00:00 0
7f8188021000-7f818c000000 ---p 00000000 00:00 0
```

我们以第一行为例:

```
00400000-00c6f000 r--p 00000000 fd:00 307653646                          /soft/mysql_3386/my
```

00400000-00c6f000 表示内存使用的范围为(16进制): 00400000 --> 00c6f000
r-p 表示权限. 具体的含义如下:

```
r = read
w = write
x = execute
s = shared
p = private (copy on write)
```

00000000 表示offset. 对于进程来讲, 使用的内存应该是连续的, 而实际分配的内存是断断续续的. 所以就使用offset来表示内存的相对位置, 这样每个进程看到的内存都是连续的了.
比如第一块内存是0x00c6f000 -> 0x00400000, 占用了8843264字节, 那么第二块内存的位置就该是8843264开始, 也就是offset=8843264 (0x0086f000) 同理: 0x02a10000 - 0x00c6f000 + 0x0086f000 = 0x02610000

> 注: 该offset是相对于fd来说的

fd:00 表示dev,就是上面说的fd

307653646 表示inode,就是文件系统的inode

```
10:11:40 [root@ddcw21 8393]#ll -i /soft/mysql_3386/mysqlbase/mysql/bin/mysqld
307653646 -rwxr-xr-x 1 7161 31415 75299232 Dec 16 18:39 /soft/mysql_3386/mysqlbase/mysql/bin/mysqld
```

/soft/mysql_3386/mysqlbase/mysql/bin/mysqld 就是对应的文件了.

mysql

所以我们只需要遍历maps就可以知道mysqld进程的内存分配情况了, 然后读取mem文件对应位置的数据查找需要的数据即可.

目录

我们直接将这部分操作整理为函数. 这里仅为演示原理. 实际使用的时候见后面演示部分即可.

```
# 在内存中查找某个关键词
def find_data_in_mem(pid,key):
        keysize = len(key)
        with open(f'/proc/{pid}/maps','r') as f:
                maps = f.readlines()

        result = []
        with open(f'/proc/{pid}/mem','rb') as f:
                for line in maps:
                        addr = line.split()[0]
                        _flags = line.split()[1]
                        if _flags != 'rw-p':
                                continue
                        start_addr,stop_addr = addr.split('-')
                        start_addr = int(start_addr,16)
                        stop_addr  = int(stop_addr ,16)
                        f.seek(start_addr,0)
                        data = f.read(stop_addr-start_addr)
                        offset = 0
                        while True:
                                offset = data.find(key,offset)
                                if offset != -1:
                                        result.append([start_addr,stop_addr,offset])
                                        offset += keysize
                                else:
                                        break
        return result
```

```
...                             if offset != -1:
...                                     result.append([start_addr,stop_addr,offset])
...                                     offset += keysize
...                             else:
...                                     break
...         return result
...
>>> find_data_in_mem(12361, b'0FE8A0B9017E2374037E9B151CBB384A05E6466B')
[[85176320, 125247488, 38816521], [140298999037952, 140298999357440, 28386], [140298999037952, 140298999357440, 79540], [140300518809600, 140303562440704, 2924753256]]
>>> f = open('/proc/12361/mem','rb')
>>> f.seek(85176320,0)
85176320
>>> data = f.read(125247488 - 85176320)
>>> data[38816521-100:38816521+41]
b"\x00\x00\x00%\x00\x00\x00\x00\x00\x00\x00\x00$A$005$DX'l0\x0b_};\x17fTh^eZGqL\x1272JTncqjTRLnVha4Gb5B1P/KbzOdi0pDDoML56/wRNC\x00\x00u33\x00\x00\x00\x00\x00%\x00\x00\x00\x00\x00\x00\x00*0FE8A0B9017E2374037E9B151CBB384A05E6466B\x00"
>>> f.seek(140298999037952,0)
140298999037952
>>> data = f.read(140298999357440-140298999037952)
>>> data[28386-100:28386+41]
b'\x01%\x03u33\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x01N\x00\x00\x00\x00\x010\x010\x010\x010\x15mysql_native_password)*0FE8A0B9017E2374037E9B151CBB384A05E6466B\x01'
>>>
```

看着是不是很熟悉, 其实就是mysql.user表里面的数据, 但是mysql的认证并不是登录的时候,直接查询mysql.user里面的密码去匹配. 所以仅修改这里是没用的. 不然直接update mysql.user表修改密码不就得了, 干嘛还要 `flush privileges` 呢? 这个fllush去了哪来呢?

回顾一下之前我们讲的mysql连接协议可知, 密码验证的是二进制的,而非16进制的, 所以内存中还存在着二进制的加密密码. flush刷新的值应该就是这里. 所以我们查找的应该是二进制的密码.

```
...         return result
...
>>>
>>> find_data_in_mem(12361, bytes.fromhex('0FE8A0B9017E2374037E9B151CBB384A05E6466B'))
[[85176320, 125247488, 38790800], [85176320, 125247488, 38901432]]
>>>
>>> f = open('/proc/12361/mem','rb')
>>> f.seek(85176320,0)
85176320
>>> data = f.read(125247488-85176320)
>>> data[38790800-100:38790800+41]
b'\x00\x00\x00\x00\xe9\x07\x00\x00\x02\x00\x00\x00\x06\x00\x00\x00t\x00\x00\x00\x1c\x00\x00\x00.\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00+\x0c\x02\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\xfe\x7f\x00\x00\x08\xfbc\x07\x00\x00\x00\x00)\x00\x00\x00\x00\x00\x00\x0c\x0f\xe8\xa0\xb9\x01~#t\x03~\x9b\x15\x1c\xbb8J\x05\xe6Fk\x00\x14\x00\x00\xc3\x15\x17\x03\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

这个位置并没有用户之类的信息,所以我们得修改所有密码为该值的账号密码. 比如u1和u2的密码都是123456, 那么修改u2的密码的时候,u1的密码也会被修改. 当然我们可以查询源码, 找到具体的对应关系. 但这都是后话了.

修改内存的话, 就是打开mem时候, 使用 `r+b` 即可, 即有写的权限, 然后f.write()即可, 就不单独演示了, 直接看后面的演示.

演示

理论是非常枯燥的, 所以我们来演示瞧瞧效果. 脚本见文末或者github上.

> 注意: --user指定user@host的时候, user和host都不需要加引号

查看用户的密码

查看的原理是遍历内存,找mysql.user表里面对应的账号记录. 如果没有查询过mysql.user表, 即mysql.user表不在内存里面的话, 是无法查询用户密码的. 当然可以查询ibd文件获取密码, frm的直接hexdump -C就行, innodb的之前也提供过脚本的.

```
python3 online_modify_mysql_password.py --user u33@%
```

```
10:42:59 [root@ddcw21 ei]#python3 online_modify_mysql_password.py --user u33@%
u33@% password:*0FE8A0B9017E2374037E9B151CBB384A05E6466B  140584077492224-140584077811712:28949
10:43:03 [root@ddcw21 ei]#
```

修改用户密码(方法1)

我们只是修改的flush处的密码, 所以如果再次flush的话, 我们修改的密码就失效了. 而且我们修改的是所有密码和u33一样的账号的, 所以我们还得登录数据库, 使用alter修改密码, 然后flush privileges刷新其它和u33密码一样的无辜者.

```
python3 online_modify_mysql_password.py --user u33@% --password newpassword_u33
```

```
10:45:46 [root@ddcw21 ei]#mysql -h127.0.0.1 -P3386 -uu33 -pu33 -e 'select current_user()'
mysql: [Warning] Using a password on the command line interface can be insecure.
+----------------+
| current_user() |
+----------------+
| u33@%          |
+----------------+
10:45:52 [root@ddcw21 ei]#python3 online_modify_mysql_password.py --user u33@% --password newpassword_u3
u33@% password:*0FE8A0B9017E2374037E9B151CBB384A05E6466B  140584077492224-140584077811712:28949
[253, 127, 0, 0, 216, 88, 4, 7, 0, 0, 0, 0, 41, 0, 0, 0, 0, 0, 0, 0]
[253, 127, 0, 0, 216, 88, 4, 7, 0, 0, 0, 0, 41, 0, 0, 0, 0, 0, 0, 0]
[253, 127, 0, 0, 224, 167, 0, 60, 220, 127, 0, 0, 41, 0, 0, 0, 0, 0, 0, 0]
[253, 127, 0, 0, 224, 167, 0, 60, 220, 127, 0, 0, 41, 0, 0, 0, 0, 0, 0, 0]
set new password succuss! (0758aa95102528f1d40ea40490526c450ef3098c)
10:46:18 [root@ddcw21 ei]#
10:46:22 [root@ddcw21 ei]#mysql -h127.0.0.1 -P3386 -uu33 -pu33 -e 'select current_user()'
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 1045 (28000): Access denied for user 'u33'@'localhost' (using password: YES)
10:46:23 [root@ddcw21 ei]#
10:46:26 [root@ddcw21 ei]#mysql -h127.0.0.1 -P3386 -uu33 -pnewpassword_u33 -e 'select current_user()'
mysql: [Warning] Using a password on the command line interface can be insecure.
+----------------+
| current_user() |
+----------------+
| u33@%          |
+----------------+
10:46:31 [root@ddcw21 ei]#
```

修改用户密码(方法2)

还有种情况就是mysql.user不在内存中, 或者flush处的密码和mysql.user的不一致(比如使用update修改密码), 那么我们就需要人工提供mysql.user里面的密码(其实是flush处的密码).

```
python3 online_modify_mysql_password.py --user u33@% --password newpassword_u33 --old-passw
```

```
10:53:35 [root@ddcw21 ei]#
10:53:36 [root@ddcw21 ei]#python3 online_modify_mysql_password.py --user u33@% --password newpassword_u33
没找到...
10:53:48 [root@ddcw21 ei]#python3 online_modify_mysql_password.py --user u33@% --password newpassword_u33 --old-password 0FE8A
0B9017E2374037E9B151CBB384A05E6466B
[253, 127, 0, 0, 88, 26, 202, 6, 0, 0, 0, 0, 41, 0, 0, 0, 0, 0, 0, 0]
[253, 127, 0, 0, 88, 26, 202, 6, 0, 0, 0, 0, 41, 0, 0, 0, 0, 0, 0, 0]
set new password succuss! (0758aa95102528f1d40ea40490526c450ef3098c)
10:54:05 [root@ddcw21 ei]#mysql -h127.0.0.1 -P3386 -uu33 -pnewpassword_u33 -e 'select current_user()'
mysql: [Warning] Using a password on the command line interface can be insecure.
+----------------+
| current_user() |
+----------------+
| u33@%          |
+----------------+
10:54:15 [root@ddcw21 ei]#
```

存在多个mysqld进程的时候

如果服务器上存在多个mysqld进程, 则需要使用 `--pid PID` 执行实际要修改的账号所在实例的进程号.

```
python3 online_modify_mysql_password.py --user u33@% --password newpassword_u33 --pid 18721
```

```
10:59:09 [root@ddcw21 ei]#python3 online_modify_mysql_password.py --user u33@% --password newpassword_u33
当前存在多个mysqld进程, 请指定一个
10:59:11 [root@ddcw21 ei]#
10:59:18 [root@ddcw21 ei]#python3 online_modify_mysql_password.py --user u33@% --password newpassword_u33 --pid 18721
u33@% password:*0FE8A0B9017E2374037E9B151CBB384A05E6466B  140092773498880-140092773834752:28981
[252, 127, 0, 0, 232, 155, 249, 7, 0, 0, 0, 0, 41, 0, 0, 0, 0, 0, 0, 0]
[252, 127, 0, 0, 232, 155, 249, 7, 0, 0, 0, 0, 41, 0, 0, 0, 0, 0, 0, 0]
set new password succuss! (0758aa95102528f1d40ea40490526c450ef3098c)
10:59:25 [root@ddcw21 ei]#
```

总结

虽然本文提供了不需要重启数据库就能强制修改密码的方法, 但还是建议重启数据库(还能释放下内存). 目前测试了5.7.38 8.0.28 8.0.41 均成功了的. 目前**仅支持mysql_native_password插件**的密码.

如果使用本脚本修改密码后,未登录数据库,做alter和flush的话, 再次使用脚本时也需要加上–old-password

参考:
https://www.kernel.org/doc/html/latest/filesystems/proc.html

源码

https://github.com/ddcw/ddcw/tree/master/python/online_modify_mysql_password

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# writen by ddcw @https://github.com/ddcw
# 在线修改mysql密码的工具. 仅支持 mysql_native_password 插件的

import os
import sys
import struct
import hashlib
import binascii
import argparse

def _argparse():
        parser = argparse.ArgumentParser(add_help=False, description='在线修改mysqld进程的脚本
        parser.add_argument('--help', '-h', action='store_true', dest="HELP", default=False
        parser.add_argument('--password', '-p', dest="PASSWORD",  help='mysql new password'
        parser.add_argument('--old-password', dest="OLD_PASSWORD",  help='last modify passw
        parser.add_argument('--pid', dest="PID", help='mysql pid', type=int)
        parser.add_argument('--user', dest="USER", help='mysql account (user@host, root@loc
        if parser.parse_args().HELP:
                parser.print_help()
                print('Example:')
                print(f'python3 f{sys.argv[0]} --user root@localhost')
                print(f'python3 f{sys.argv[0]} --user root@localhost --password 123456')
                print(f'python3 f{sys.argv[0]} --user root@localhost --password 123456 --pi
                sys.exit(0)
        if parser.parse_args().USER is None:
                print('必须使用 --user 指定用户')
                sys.exit(10)
        return parser.parse_args()

def encode_password(NEW_PASSWORD):
        return hashlib.sha1(hashlib.sha1(NEW_PASSWORD.encode()).digest()).digest()
```

```python
# 在内存中查找某个关键词
def find_data_in_mem(pid,key):
        keysize = len(key)
        with open(f'/proc/{pid}/maps','r') as f:
                maps = f.readlines()

        result = []
        with open(f'/proc/{pid}/mem','rb') as f:
                for line in maps:
                        addr = line.split()[0]
                        _flags = line.split()[1]
                        if _flags != 'rw-p':
                                continue
                        start_addr,stop_addr = addr.split('-')
                        start_addr = int(start_addr,16)
                        stop_addr  = int(stop_addr ,16)
                        f.seek(start_addr,0)
                        data = f.read(stop_addr-start_addr)
                        offset = 0
                        while True:
                                offset = data.find(key,offset)
                                if offset != -1:
                                        result.append([start_addr,stop_addr,offset])
                                        offset += keysize
                                else:
                                        break
        return result

# 设置新密码, 是直接将旧密码改为新密码, 如果多个用户的密码是一样的, 则都会修改, 不修改mysql.user等信息
def set_new_password(OLD_PASSWORD,NEW_PASSWORD,pid):
        maps = find_data_in_mem(pid,OLD_PASSWORD)
        if len(maps) == 0:
                print('可能之前已经修改过了, 可以使用--old-password 指定上一次的密码')
                sys.exit(1)
        with open(f'/proc/{pid}/mem','r+b') as f:
                for start,stop,offset in maps:
                        f.seek(start+offset-20,0)
                        data = f.read(20)
                        if data[-4:] != b'\x00\x00\x00\x00':
                                continue
                        # 5.7 255, 4, 0, 0, 0, 0
                        # 8.0 41, 0, 0, 0, 0, 0, 0, 0
                        print([ x for x in data ])
                        f.seek(start+offset,0)
                        f.write(NEW_PASSWORD)
        print(f'set new password succuss! ({binascii.hexlify(NEW_PASSWORD).decode()})')


def get_pid(): # 获取mysqld进程的pid
        pid = []
        for entry in os.listdir('/proc'):
                if not entry.isdigit():
                        continue
                try:
                        comm = '/proc/'+str(entry)+'/comm'
                        with open(comm,'r') as f:
                                if f.read() == 'mysqld\n':
                                        pid.append(entry)
                except:
                        pass
        return pid


if __name__ == "__main__":
        parser = _argparse()
        user,host = parser.USER.split('@')
        flags = struct.pack('<B',len(host)) + host.encode() + struct.pack('<B',len(user)) +
        PIDS = get_pid()
        pid = 0
```

```
        if parser.PID is not None:
                if str(parser.PID) in PIDS:
                        pid = parser.PID
                else:
                        print(f'pid:{parser.PID} not exists {PIDS}')
                        sys.exit(0)
        elif len(PIDS) == 1:
                pid = PIDS[0]
        elif len(PIDS) == 0:
                print('当前不存在mysqld进程')
                sys.exit(2)
        else:
                print(f'当前存在多个mysqld进程, 请指定一个')
                sys.exit(3)
MODIFY_PASSWORD = False # 是否要修改密码, 如果没有指定密码, 则仅查看即可. 若指定了密码, 则为:
NEW_PASSWORD = b''
if parser.PASSWORD is not None:
        NEW_PASSWORD = encode_password(parser.PASSWORD)
        MODIFY_PASSWORD = True
if parser.OLD_PASSWORD is not None:
        set_new_password(bytes.fromhex(parser.OLD_PASSWORD),NEW_PASSWORD,pid,)
        sys.exit(0)
# 查看当前的密码
maps = find_data_in_mem(pid,flags)
if len(maps) == 0:
        print('没找到...')
        sys.exit(1)
with open(f'/proc/{pid}/mem','rb') as f:
        for start,stop,offset in maps:
                f.seek(start,0)
                data = f.read(stop-start)
                MATCHED = True
                offset += len(flags)
                for i in range(29): # 29个权限
                        if data[offset:offset+1] != b'\x01':
                                MATCHED = False
                                break
                        else:
                                offset += 2
if not MATCHED:
        continue
# 然后就是ssl,max_conn之类的信息
for i in range(8):
        vsize = struct.unpack('<B',data[offset:offset+1])[0]
        offset += 1 + vsize
# 然后就是mysql_native_password了
vsize = struct.unpack('<B',data[offset:offset+1])[0]
plugins = data[offset+1:offset+1+vsize].decode()
offset += 1 + vsize
if plugins != 'mysql_native_password':
        continue
# 最后就是密码(password_expired之类的就不管了. 没必要)
vsize = struct.unpack('<B',data[offset:offset+1])[0] # 肯定得是41, 就
old_password = data[offset+1:offset+1+vsize].decode()
print(f'{parser.USER} password:{old_password}  {start}-{stop}:{offs
if MODIFY_PASSWORD: # 要修改的密码实际上是二进制的, 修改page的是没用的
        set_new_password(bytes.fromhex(old_password[1:]),NEW_PASSWC
        # mysql.user也修改下, 不然再次修改的时候,就找不到位置了. 算逑!
        #with open(f'/proc/{pid}/mem','r+b') as fw:
        #       fw.seek(start+offset+1,0)
        #       fw.write(NEW_PASSWORD)
break
```

墨力计划 mysql linux python

「喜欢这篇文章，您的关注和赞赏是给作者最好的鼓励」

关注作者 赞赏

1人已赞赏

文章被以下合辑收录

PYTHON解析MYSQL（共59篇）

用python解析mysql协议,模拟mysql连接,解析binlog,redolog,ibd文件等

收藏合辑

评论

分享你的看法，一起交流吧~

相关阅读

ACDU周度精选 | 本周数据库圈热点 + 技术干货分享（2025/7/25期）

墨天轮小助手 473次阅读 2025-07-25 15:54:18

ACDU周度精选 | 本周数据库圈热点 + 技术干货分享（2025/7/17期）

墨天轮小助手 437次阅读 2025-07-17 15:31:18

墨天轮「实操看我的」数据库主题征文活动启动

墨天轮编辑部 391次阅读 2025-07-22 16:11:27

深度解析MySQL的半连接转换

听见风的声音 207次阅读 2025-07-14 10:23:00

MySQL 9.4.0 正式发布，支持 RHEL 10 和 Oracle Linux 10

严少安 206次阅读 2025-07-23 01:21:32

索引条件下推和分区-一条SQL语句执行计划的分析

听见风的声音 198次阅读 2025-07-23 09:22:58

null和子查询--not in和not exists怎么选择？

听见风的声音 182次阅读 2025-07-21 08:54:19

MySQL数据库SQL优化案例(走错索引)

陈举超 167次阅读 2025-07-17 21:24:40

使用 MySQL Clone 插件为MGR集群添加节点

黄山谷 166次阅读 2025-07-23 22:04:19

7月“墨力原创作者计划”获奖名单公布

墨天轮编辑部 147次阅读 2025-08-12 15:05:56