42 | grant之后要跟着flush privileges吗

便于说明，先创建一个用户

create user 'ua'@'%' identified by 'pa';

这条语句的逻辑是创建一个用户’ua’@’%’,密码是pa，注意，在MySQL里面，用户名（user）+地址（host）才表示一个用户，因此ua@ip1和ua@ip2代表的是两个不同的用户

这条命令做了两个操作

1. 在磁盘上，往mysql.user遍历插入一行，由于没有指定权限，所以这行数据上所有表示权限的字段的值都是N
2. 内存里，往数组acl\_users里插入一个acl\_user对象，这个对象的access字段值为0。

图1，就是这个用户此时ua在user表中的状态

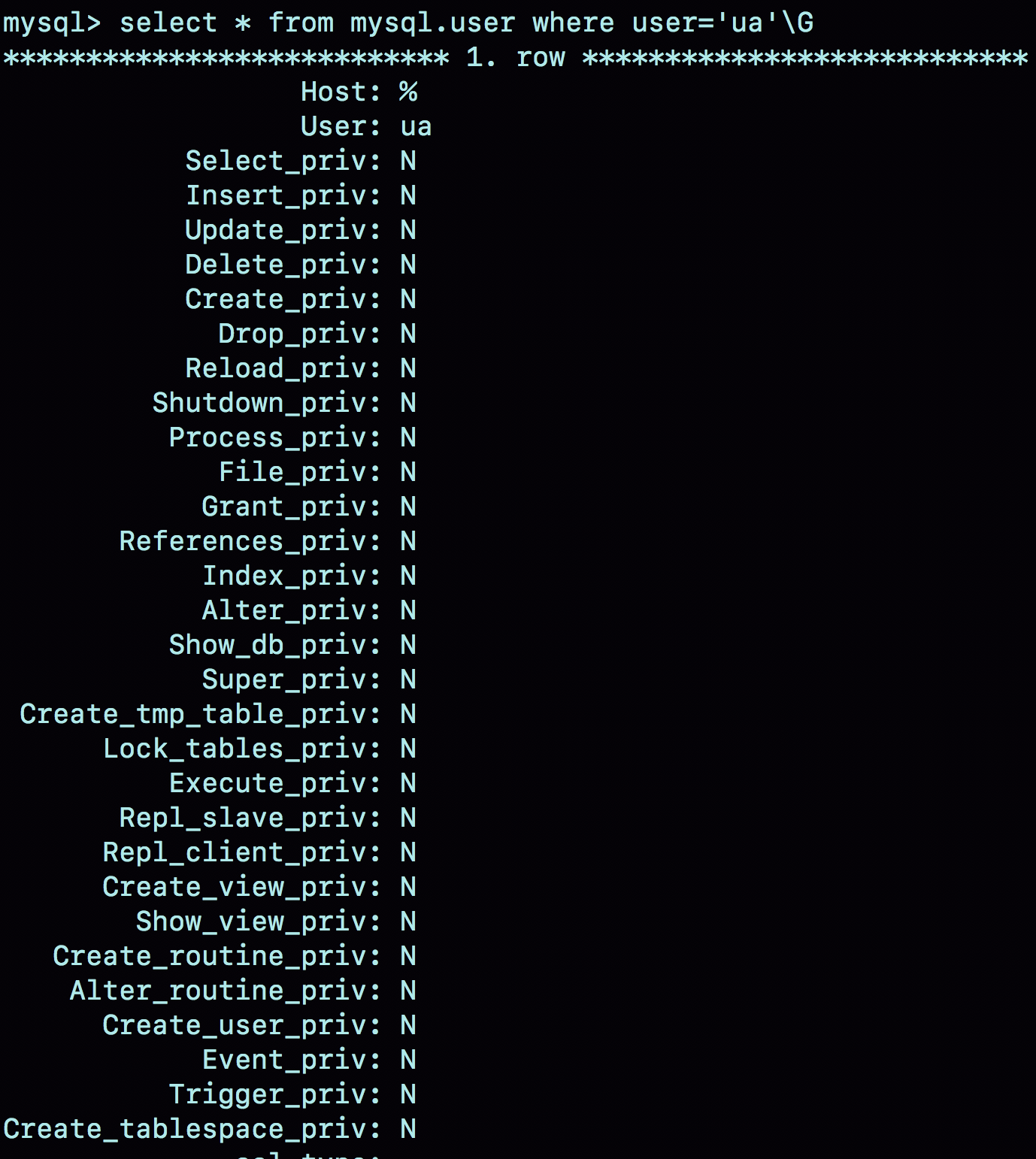


图 1 mysql.user数据行

在MySQL中，用户权限是有不同范围的，接下来，按照用户权限范围从大到小的顺序依次介绍

**全局权限**

全局权限，作用于整个MySQL实例，这些权限信息保存在mysql库的user表里，如果要给用户ua赋一个最高权限的话，语句是这么写的

grant all privileges on \*.\* to 'ua'@'%' with grant option;

这个grant命令做了两个动作；

1. 磁盘上，将mysql.user表里，用户’ua’@’%这一行所有表示权限的字段的值都修改为Y
2. 内存里，从数组acl\_users中找到这个用户对象，将access值（权限位）修改为二进制的全1

在这个grant命令执行完，如果有新的客户端使用用户ua登录成功，MySQL会为新连接维护一个线程对象，然后从acl\_users数组里查到这个用户的权限，并将权限值拷贝到这个线程对象中。之后这个链接执行的语句，所有关于全局权限的判断，都直接使用线程对象内部保存的权限位。

基于上面的分析，得知

1. grant命令对于全局权限，同时更新了磁盘和内存。命令完成后即时生效，接下来新创建的连接会使用新的权限
2. 对于一个已经存在的连接，他的全局权限不受grant命令的影响。

需要说明的是，**一般生产上要合理控制用户权限的范围**，上面用到的这个grant语句是一个典型的错误示范，如果一个用户有所有权限，一般就不应该设置为所有ip地址都可以访问

如果要回收上面的grant语句赋予的权限，使用下面的命令

revoke all privileges on \*.\* from 'ua'@'%';

这条revoke命令的用法与grang类似，做了如下连个动作

1. 磁盘上，将mysql.user表里，用户’ua’@’%这一行的所有表示权限的的字段值都修改为N
2. 内存里，从数组acl\_users中找到这个用户对应的对象，将access的值修改为0.

**db权限**

除了全局权限， MySQL也支持库级别的权限定义，如果要让用户ua拥有库db1的所有权限，可以执行下面的命令

grant all privileges on db1.\* to 'ua'@'%' with grant option;

基于库的权限记录保存在mysql.db表中，在内存里则保存在数组acl\_dbs中，这个条grant命令做了如下两个动作；

1. 磁盘上，往mysql.db表中插入了一行记录，所有权限位字段设置为Y
2. 内存里，增加一个对象到数组acl\_dbs中，这个对象的权限位为全1

图2是这个用户ua在db表中的状态



图2 mysql.db 数据行

每次需要判断一个用户对应一个数据库读写权限的时，都需要遍历一次acl\_dbs数组，根据user、host和db找到匹配的对象，然后根据对应的权限位来判断。

也即是说，grant修改db权限的时候，是同时对磁盘和内存生效的

grant操作对于已经存在的连接的影响，在全局权限和基于db的权限效果是不同的，接下来对比试验分析



图3 权限操作效果

虽然用户ua的super权限在T3时刻已经通过revoke语句回收了，但是在T4时刻执行set global的时候，权限验证还是通过了。这是因为super是全局权限，这个权限信息在线程对象中，而revoke操作影响不到这个线程对象。

而在T5时刻去掉ua对db1库的所有权限后，在T6时刻sessonB 在操作db1库的表，就会报“权限不足”。这是因为acl\_dbs是一个全局数组，所有线程判断db权限都用这个数组，这样revoke操作马上就会影响到sessionB.

这里在代码上实现上有个特别的逻辑，如果当前会话已经处于某一个db里面，之前use这个库时候拿到的库权限会保存在会话变量中

可以看到T6时刻，sessionC和sessionB对表t的操作逻辑是一样的，但是sessionB报错，而sessionC在T2时刻执行的use db1，拿到了这个库的权限，在切换db1之前，sessionC对这个库就就一直有权限。

**表权限和列权限**

除了db级别的权限外，MySQL支持跟细粒度的表权限和列权限，其中，表权限定义存放在表mysql.table\_priv中，列权限定义存放在mysql.column\_priv中。这两类权限，组合起来存放在内存的hash结构column\_priv\_hash中

这两类权限命令如下

create table db1.t1(id int, a int);

grant all privileges on db1.t1 to 'ua'@'%' with grant option;

GRANT SELECT(id), INSERT (id,a) ON mydb.mytbl TO 'ua'@'%' with grant option;

跟db权限类似，这个两个权限每次grant时都会修改数据表，也会同步修改内存中的hash结构，因此对这两类权限的操作，也会马上影响到已经存在的连接。

看到这里，疑问：看来grant语句都是即时生效，这么看应该就不需要flush privileges语句了啊。答案：确实是这样的

Flush privileges命令清空acl\_users数组，然后从mysql.user表中读取数据重新加载，重新构造一个acl\_users数组，也就是说，以数据表中的数据为准，会将全局权限内存数组重新加载一遍。

同样的，对于db权限、表权限和列权限，MySQL也做了这样的处理。

也就是说，如果内存的权限数据和磁盘数据表相同的话，不需要执行flush privileges。而如果都是用grant/revoke语句来执行的话，内存和数据表本来就是保持同步更新的。

**因此，正常情况下，grant命令之后，没有必要跟着执行flush privileges命令**

**Flush privileges**

Flush privileges是在什么场景使用呢？显然，当数据库表中的权限数据跟内存中的权限数据不一致的时候，flush privileges语句可以用来重建内存数据，达到一致状态

这种不一致的往往是由不规范的操作导致的，比如直接用DML语句操作系统权限表，看看下面的场景

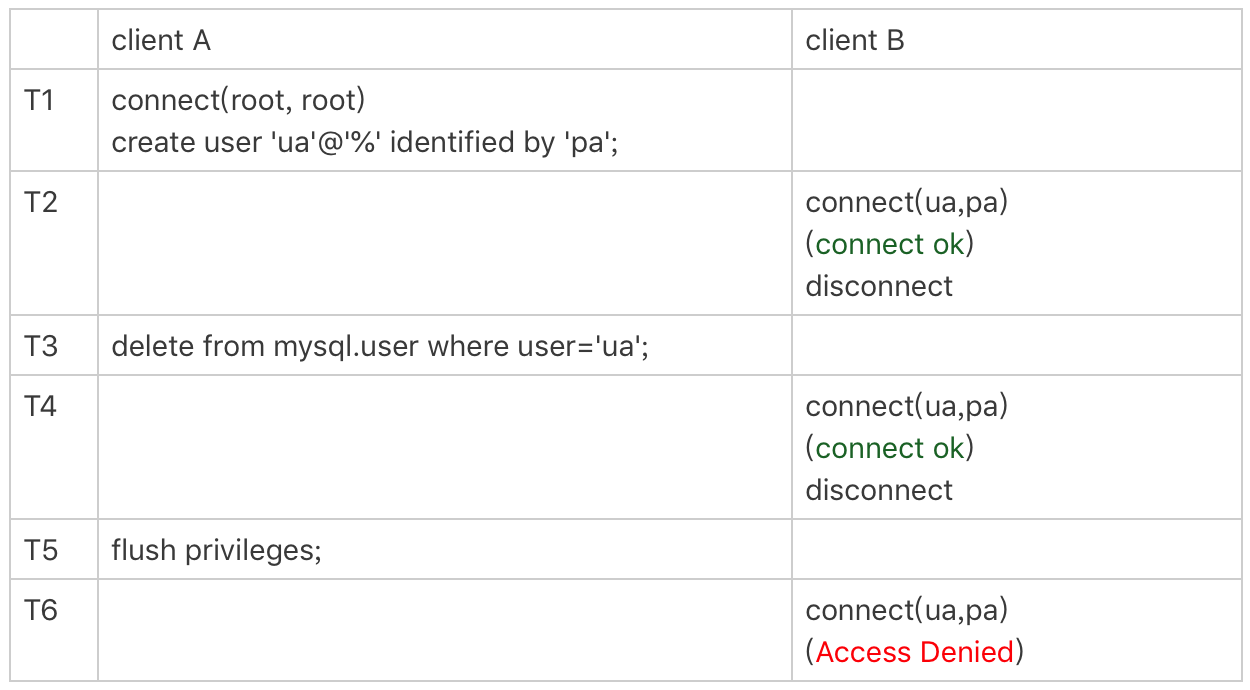


图 4 使用flush privileges

T3时刻虽然已经用delete语句删除了用户ua,但是在T4时刻，仍然可以用ua连接成功，原因在于，此时内存中acl\_users数组中还有这个用户，因此系统判断时认为用户正常存在。

在T5时刻执行过flush 命令后，内存更新，T6时刻在用ua来登录的话，就会报“无法访问”

直接操作系统表是不规范的操作，这个不一致状态会导致一些更“诡异”的现象发生，比如，前面这个通过delete语句删除用户的例子，就会出现下面的情况

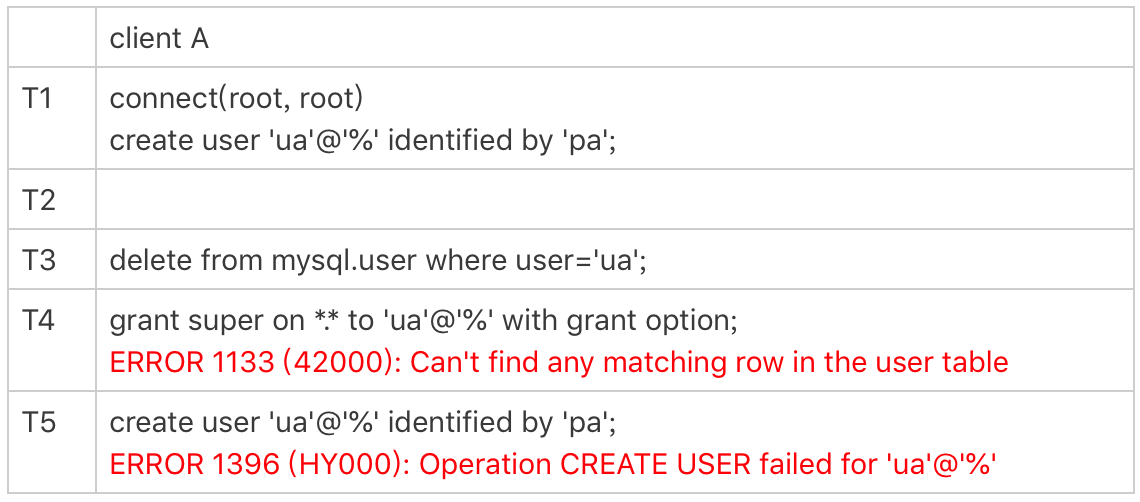


图 5 不规范权限操作导致的异常

T3时刻，直接删除了数据表的记录，而内存的数据还在，这就导致了

1. T4时刻给用户ua赋权限失效，因为mysql.user表中找不到这行记录
2. 而T5时刻用重新创建这个用户也不行，因为在做内存判断时候，会认为这个用户还存在

不推荐的使用方法

grant super on \*.\* to 'ua'@'%' identified by 'pa';

这条命令加了identified by ‘密码’语句的逻辑除了赋权限外，还包含了

1. 如果用户不存在，就户创建这个用户，密码是pa
2. 如果用户存在，就将密码修改为pa

这种写法容易将密码修改