

第03章_用户与权限管理

讲师：尚硅谷-宋红康（江湖人称：康师傅）

官网：<http://www.atguigu.com>

1.用户管理

MySQL用户可以分为普通用户和root用户。root用户是超级管理员，拥有所有权限，包括创建用户、删除用户和修改用户的密码等管理权限；普通用户只拥有被授予的各种权限。

MySQL提供了许多语句用来管理用户账号，这些语句可以用来管理包括登录和退出MySQL服务器、创建用户、删除用户、密码管理和权限管理等内容。

MySQL数据库的安全性需要通过账户管理来保证。

1.1登录MySQL服务器

启动MySQL服务后，可以通过mysql命令来登录MySQL服务器，命令如下：

```
1 | mysql -h hostname|hostIP -P port -u username -p DatabaseName -e "SQL语句"
```

下面详细介绍命令中的参数：

- `-h`参数 后面接主机名或者主机IP，hostname为主机，hostIP为主机IP。
- `-P`参数 后面接MySQL服务的端口，通过该参数连接到指定的端口。MySQL服务的默认端口是3306，不使用该参数时自动连接到3306端口，port为连接的端口号。
- `-u`参数 后面接用户名，username为用户名。
- `-p`参数 会提示输入密码。
- `DatabaseName`参数 指明登录到哪一个数据库中。如果没有该参数，就会直接登录到MySQL数据库中，然后可以使用USE命令来选择数据库。
- `-e`参数 后面可以直接加SQL语句。登录MySQL服务器以后即可执行这个SQL语句，然后退出MySQL服务器。

举例：

```
1 | mysql -uroot -p -hlocalhost -P3306 mysql -e "select host,user from user"
```

1.2创建用户

在MySQL数据库中，官方推荐使用 `CREATE USER` 语句创建新用户。MySQL 8版本移除了PASSWORD加密方法，因此不再推荐使用INSERT语句直接操作MySQL中的user表来增加用户。

使用CREATE USER语句来创建新用户时，**必须拥有CREATE USER权限**。每添加一个用户，CREATE USER语句会在MySQL.user表中添加一条新记录，但是**新创建的账户没有任何权限**。如果添加的账户已经存在，CREATE USER语句就会返回一个错误。

CREATE USER语句的基本语法形式如下：

```
1 CREATE USER 用户名 [IDENTIFIED BY '密码'][,用户名 [IDENTIFIED BY '密码']];
```

- 用户名参数表示新建用户的账户，由 用户（User）和 主机名（Host）构成；
- “[]”表示可选，也就是说，可以指定用户登录时需要密码验证，也可以不指定密码验证，这样用户可以直接登录。不过，不指定密码的方式不安全，不推荐使用。如果指定密码值，这里需要使用 IDENTIFIED BY 指定明文密码值。
- CREATE USER 语句可以同时创建多个用户。

举例：

```
1 CREATE USER zhang3 IDENTIFIED BY '123123'; # 默认host是 %
```

```
1 CREATE USER 'kangshifu'@'localhost' IDENTIFIED BY '123456';
```

1.3 修改用户

修改用户名：

```
1 UPDATE mysql.user SET USER='li4' WHERE USER='wang5';
2 FLUSH PRIVILEGES;
```

```
mysql> update mysql.user set user='li4' where user='wang5';
Query OK, 1 row affected (0.27 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

1.4 删除用户

在MySQL数据库中，可以使用 DROP USER 语句来删除普通用户，也可以直接在mysql.user表中删除用户。

方式1：使用DROP方式删除（推荐）

使用DROP USER语句来删除用户时，必须用于 DROP USER 权限。DROP USER 语句的基本语法形式如下：

```
1 DROP USER user[,user...];
```

其中，user参数是需要删除的用户，由用户的用户名（User）和主机名（Host）组成。DROP USER 语句可以同时删除多个用户，各用户之间用逗号隔开。

举例：

```
1 DROP USER li4 ; # 默认删除host为%的用户
```

```
1 DROP USER 'kangshifu'@'localhost';
```

```
mysql> drop user li4;
Query OK, 0 rows affected (0.00 sec)

mysql> select host,user,password,select_priv,insert_priv,drop_priv from mysql.user;
+-----+-----+-----+-----+-----+-----+
| host      | user | password                                     | select_priv | insert_priv | drop_priv |
+-----+-----+-----+-----+-----+-----+
| %         | root | *E56A114692FE0DE073F9A1DD68A00EEB9703F3F1 | Y           | Y           | Y         |
| jack.atguigu | root |                                               | Y           | Y           | Y         |
| 127.0.0.1 | root |                                               | Y           | Y           | Y         |
| ::1       | root |                                               | Y           | Y           | Y         |
| localhost |      |                                               | N           | N           | N         |
| jack.atguigu |      |                                               | N           | N           | N         |
| localhost | root | *E56A114692FE0DE073F9A1DD68A00EEB9703F3F1 | Y           | Y           | Y         |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

方式2：使用DELETE方式删除

可以使用DELETE语句直接将用户的信息从mysql.user表中删除，但必须拥有对mysql.user表的DELETE权限，DELETE语句的基本语法形式如下：

```
1 | DELETE FROM mysql.user WHERE Host='hostname' AND User='username';
```

Host字段和User字段是user表的联合主键，因此**两个字段的值才能唯一确定一条记录**。

执行完DELETE命令后要使用FLUSH命令来使用户生效，命令如下：

```
1 | FLUSH PRIVILEGES;
```

举例：

```
1 | DELETE FROM mysql.user WHERE Host='localhost' AND User='Emily';
2 | FLUSH PRIVILEGES;
```

注意：不推荐通过 `DELETE FROM USER u WHERE USER='li4'` 进行删除，系统会有残留信息保留。而drop user命令会删除用户以及对应的权限，执行命令后你会发现 mysql.user表和mysql.db表的相应记录都消失了。

1.5设置当前用户密码

适用于root用户修改自己的密码，以及普通用户登录后修改自己的密码。

root用户拥有很高的权限，因此必须保证root用户的密码安全。root用户可以通过多种方式来修改密码，使用 `ALTER USER` 修改用户密码是MySQL官方推荐的方式。此外，也可以通过 `SET` 语句修改密码。由于MySQL8中已移除了PASSWORD() 函数，因此不再使用UPDATE语句直接操作用户表修改密码。

旧的写法如下：

```
1 | #修改当前用户的密码：（MySQL5.7测试有效）
2 | SET PASSWORD = PASSWORD('123456');
```

这里介绍**推荐的写法**：

1.使用ALTER USER命令来修改当前用户密码

用户可以使用ALTER命令来修改自身密码，如下语句代表修改当前登录用户的密码。基本语法如下：

```
1 | ALTER USER USER() IDENTIFIED BY 'new_password';
```

练习：下面使用ALTER命令来修改root用户的密码，将密码改为“Hello_1234”。命令如下：

```
1 ALTER USER USER() IDENTIFIED BY 'Hello_1234';
```

2.使用SET语句来修改当前用户密码

使用root用户登录MySQL后，可以使用SET语句来修改密码，具体 SQL语句如下：

```
1 SET PASSWORD='new_password';
```

该语句会自动将密码加密后再赋给当前用户。

练习：下面使用SET语句来修改root用户的密码，将密码改为“Hello_1234”。SET语句具体如下：

```
1 SET PASSWORD='Hello_1234';
```

1.6修改其它用户密码

root用户不仅可以修改自己的密码，还可以修改其它普通用户的密码。root用户登录MySQL服务器后，可以通过 ALTER语句 和 SET语句 来修改普通用户的密码。由于PASSWORD（函数已移除，因此使用UPDATE直接操作用户表的方式已不再使用

1.使用ALTER语句来修改普通用户的密码

可以使用ALTERUSER语句来修改普通用户的密码。基本语法形式如下：

```
1 ALTER USER user [IDENTIFIED BY '新密码 ']  
2 [,user[IDENTIFIED BY '新密码 ']]...;
```

其中，user参数表示新用户的账户，由用户名和主机名构成；“IDENTIFIEDBY”关键字用来设置密码。

练习：下面使用ALTER语句来修改kangshifu用户的密码，将密码改为“HelloWorld_123”。

```
1 ALTER USER 'kangshifu' @ 'localhost' IDENTIFIED BY 'HelloWorld_123';
```

2.使用SET命令来修改普通用户的密码

使用root用户登录到MySQL服务器后，可以使用SET语句来修改普通用户的密码。SET语句的代码如下：

```
1 SET PASSWORD FOR 'username' @ 'hostname'='new_password';
```

其中，username参数是普通用户的用户名；hostname参数是普通用户的主机名；new_password是新密码。

练习：下面使用SET语句来修改kangshifu用户的密码，将密码改成“HelloWorld_123”。

```
1 SET PASSWORD for 'kangshifu'@'localhost'='HelloWorld_123';
```

3.使用UPDATE语句修改普通用户的密码（不推荐）

使用root用户登录MySQL服务器后，可以使用UPDATE语句修改MySQL数据库的user表的password字段，从而修改普通用户的密码。使用UPDATA语句修改用户密码的语法如下：

```
1 UPDATE MySQL.user SET authentication_string=PASSWORD("123456")  
2 WHERE User = "username" AND Host = "hostname";
```

举例：

```
1 #修改某个用户的密码：
2 #MySQL5.5
3 UPDATE mysql.user SET PASSWORD=PASSWORD('123456') WHERE USER='li4';
4
5 #MySQL5.7（不适用于MySQL8）
6 UPDATE mysql.user SET authentication_string=PASSWORD('123456') WHERE
  USER='li4';
7
8 FLUSH PRIVILEGES; #所有通过user表的修改，必须用该命令才能生效。否则。需重启服务。
```

PASSWORD() 函数用来加密用户密码。执行UPDATE语句后，需要执行 FLUSH PRIVILEGE 语句重新加载用户权限。

1.7MySQL8密码管理(了解)

MySQL中记录使用过的历史密码，目前包含如下密码管理功能：

1. 密码过期：要求定期修改密码
2. 密码重用限制：不允许使用旧密码
3. 密码强度评估：要求使用高强度的密码（第1章已讲）

提示

MySQL密码管理功能只针对使用基于MySQL授权插件的账号，这些插件有 mysql_native_password、sha256_password 和 caching_sha2_password。

1.密码过期策略

- 在MySQL中，数据库管理员可以手动设置账号密码过期，也可以建立一个自动密码过期策略。
- 过期策略可以是全局的，也可以为每个账号设置单独的过期策略。

手动设置立马过期

手动设置账号密码过期，可使用如下语句：

```
1 ALTER USER user PASSWORD EXPIRE;
```

练习：

```
1 ALTER USER 'kangshifu'@'localhost' PASSWORD EXPIRE;
```

该语句将用户kangshifu的密码设置为过期，kangshifu用户仍然可以登录进入数据库，但无法进行查询。密码过期后，只有重新设置了新密码，才能正常使用。

```
mysql> show databases;
ERROR 1820 (HY000): You must reset your password using ALTER USER statement before executing this statement.
```

手动设置指定时间过期方式1：全局

如果密码使用的时间大于允许的时间，服务器会自动设置为过期，不需要手动设置。

MySQL使用 default_password_lifetime 系统变量建立全局密码过期策略。

- 它的默认值是0，表示禁用自动密码过期。

- 它允许的值是正整数N，表示允许的密码生存期。密码必须 每隔N天 进行修改。

两种实现方式分别如图所示：

- 方式①：使用SQL语句更改该变量的值并持久化

```
1 SET PERSIST default_password_lifetime = 180; #建立全局策略，设置密码每隔 180天过期
```

```
mysql> SET PERSIST default_password_lifetime = 180;
Query OK, 0 rows affected (0.00 sec)
```

- 方式②：配置文件my.cnf中进行维护

```
1 [mysqld]
2 default_password_lifetime=180 #建立全局策略，设置密码每隔180天过期
```

手动设置指定时间过期方式2：单独设置

每个账号既可延用全局密码过期策略，也可单独设置策略。在 `CREATE USER` 和 `ALTER USER` 语句上加入 `PASSWORD EXPIRE` 选项可实现单独设置策略。下面是一些语句示例。

```
1 #设置 kangshifu账号密码每 90天过期：
2 CREATE USER 'kangshifu'@'localhost' PASSWORD EXPIRE INTERVAL 90 DAY;
3 ALTER USER 'kangshifu'@'localhost' PASSWORD EXPIRE INTERVAL 90 DAY;
4
5 #设置密码永不过期：
6 CREATE USER 'kangshifu'@'localhost' PASSWORD EXPIRE NEVER;
7 ALTER USER 'kangshifu'@'localhost' PASSWORD EXPIRE NEVER;
8
9 #延用全局密码过期策略：
10 CREATE USER 'kangshifu'@'localhost' PASSWORD EXPIRE DEFAULT;
11 ALTER USER 'kangshifu'@'localhost' PASSWORD EXPIRE DEFAULT;
```

2.密码重用策略

MySQL限制使用已用过的密码。重用限制策略基于 密码更改的数量 和 使用的时间 。重用策略可以是 全局 的，也可以为每个账号设置 单独的 策略 。

- 号的历史密码包含过去该账号所使用的密码。MySQL基于以下规则来限制密码重用：
 - 如果账号的密码限制 基于密码更改的数量 ，那么新密码不能从最近限制的密码数量中选择。例如，如果密码更改的最小值为3，那么新密码不能与最近3个密码中任何一个相同。
 - 如果账号密码限制 基于时间 ，那么新密码不能从规定时间内选择。例如，如果密码重用周期为60天，那么新密码不能从最近60天内使用的密码中选择。
- MySQL使用password_history和password_reuse_interval系统变量设置密码重用策略。
 - password_history：规定密码重用的数量
 - password_reuse_interval：规定密码重用的周期
- 这两个值可在 服务器的配置文件中 进行维护，也可在运行期间 使用SQL语句更改 该变量的值并持久化。
- 手动设置密码重用方式1：全局
 - 方式①：使用SQL

```
1 SET PERSIST password_history = 6; # 设置不能选择最近使用过的6个密码
2 SET PERSIST password_reuse_interval = 365; # 设置不能选择最近一年内的密码
```

```
mysql> SET PERSIST password_history = 6;
Query OK, 0 rows affected (0.00 sec)

mysql> SET PERSIST password_reuse_interval = 365;
Query OK, 0 rows affected (0.00 sec)
```

- 方式②：my.cnf配置文件

```
1 [mysqld]
2 password_history=6
3 password_reuse_interval=365
```

- 手动设置密码重用方式2：单独设置

```
1 #不能使用最近 5个密码：
2 CREATE USER 'kangshifu'@'localhost' PASSWORD HISTORY 5;
3 ALTER USER 'kangshifu'@'localhost' PASSWORD HISTORY 5;
4
5 #不能使用最近 365天内的密码：
6 CREATE USER 'kangshifu'@'localhost' PASSWORD REUSE INTERVAL 365 DAY;
7 ALTER USER 'kangshifu'@'localhost' PASSWORD REUSE INTERVAL 365 DAY;
8
9 #既不能使用最近 5个密码，也不能使用 365天内的密码
10 CREATE USER 'kangshifu'@'localhost'
11 PASSWORD HISTORY 5
12 PASSWORD REUSE INTERVAL 365 DAY;
13
14 ALTER USER 'kangshifu'@'localhost'
15 PASSWORD HISTORY 5
16 PASSWORD REUSE INTERVAL 365 DAY;
```

```
1 #延用全局策略
2 CREATE USER 'kangshifu' @ 'localhost'
3 PASSWORD HISTORY DEFAULT
4 PASSWORD REUSE INTERVAL DEFAULT;
5
6 ALTER USER 'kangshifu' @ 'localhost'
7 PASSWORD HISTORY DEFAULT
8 PASSWORD REUSE INTERVAL DEFAULT;
```

2.权限管理

关于MySQL的权限简单的理解就是MySQL允许你做你权力以内的事情，不可以越界。比如只允许你执行SELECT操作，那么你就不能执行UPDATE操作。只允许你从某台机器上连接MySQL，那么你就不能从除那台机器以外的其他机器连接MySQL。

2.1权限列表

MySQL到底都有哪些权限呢？

```
1 | mysql> show privileges;
```

GRANT和REVOKE语句中可以使用的权限如下：

权限	user 表中对应的列	权限的范围
CREATE	Create_priv	数据库、表或索引
DROP	Drop_priv	数据库、表或视图
GRANT OPTION	Grant_priv	数据库、表或存储过程
REFERENCES	References_priv	数据库或表
EVENT	Event_priv	数据库
ALTER	Alter_priv	数据库
DELETE	Delete_priv	表
INDEX	Index_priv	表
INSERT	Insert_priv	表

ALTER ROUTINE	Alter_routine_priv	存储过程和函数
CREATE ROUTINE	Create_routine_priv	存储过程和函数
EXECUTE	Execute_priv	存储过程和函数
FILE	File_priv	访问服务器上的文件
CREATE TABLESPACE	Create_tablespace_priv	服务器管理
CREATE USER	Create_user_priv	服务器管理
PROCESS	Process_priv	存储过程和函数
RELOAD	Reload_priv	访问服务器上的文件
REPLICATION CLIENT	Repl_client_priv	服务器管理
REPLICATION SLAVE	Repl_slave_priv	服务器管理
SHOW DATABASES	Show_db_priv	服务器管理
SHUTDOWN	Shutdown_priv	服务器管理
SUPER	Super_priv	服务器管理

- 1. CREATE和DROP权限，可以创建新的数据库和表，或删除（移掉）已有的数据库和表。如果将MySQL数据库中的DROP权限授予某用户，用户就可以删除MySQL访问权限保存的数据库。
- 2. SELECT、INSERT、UPDATE和DELETE权限 允许在一个数据库现有的表上实施操作。
- 3. SELECT权限 只有在它们真正从一个表中检索行时才被用到。
- 4. INDEX权限 允许创建或删除索引，INDEX适用于已有的表。如果具有某个表的CREATE权限，就可以在CREATETABLE语句中包括索引定义。
- 5. ALTER权限 可以使用ALTERTABLE来更改表的结构和重新命名表。
- 6. CREATE ROUTINE权限 用来创建保存的程序（函数和程序），ALTERROUTINE权限用来更改和删除保存的程序，EXECUTE权限用来执行保存的程序。
- 7. GRANT权限 允许授权给其他用户，可用于数据库、表和保存的程序。
- 8. FILE权限 使用户可以使用LOADDATAINFILE和SELECT ... INTOOUTFILE语句读或写服务器上的文件，任何被授予FILE权限的用户都能读或写MySQL服务器上的任何文件（说明用户可以读任何数据库目录下的文件，因为服务器可以访问这些文件）。

MySQL的权限如何分布：

权限分布	可能设置的权限
表权限	'Select','Insert','Update','Delete','Create','Drop','Grant','References','Index','Alter'
列权限	'Select','Insert','Update','References'
过程权限	'Execute','AlterRoutine','Grant'

2.2 授予权限的原则

权限控制主要是出于安全因素，因此需要遵循以下几个经验原则：

1. 只授予能满足需要的最小权限，防止用户干坏事。比如用户只是需要查询，那就只给select权限就可以了，不要给用户赋予update、insert或者delete权限。
2. 创建用户的时候限制用户的登录主机，一般是限制成指定IP或者内网IP段。
3. 为每个用户设置满足密码复杂度的密码。
4. 定期清理不需要的用户，回收权限或者删除用户。

2.3 授予权限

给用户授权的方式有 2 种，分别是通过把角色赋予用户给用户授权和直接给用户授权。用户是数据库的使用者，我们可以通过给用户授予访问数据库中资源的权限，来控制使用者对数据库的访问，消除安全隐患。

授权命令：

```
1 GRANT 权限1,权限2,...权限n ON 数据库名称.表名称 TO 用户名@用户地址 [IDENTIFIED BY '密码口令'];
```

- 该权限如果发现没有该用户，则会直接新建一个用户。

比如：

- 给li4用户用本地命令行方式，授予atguigudb这个库下的所有表的插删改查的权限。

```
1 GRANT SELECT,INSERT,DELETE,UPDATE ON atguigudb.* TO li4@localhost ;
```

- 授予通过网络方式登录的joe用户，对所有库所有表的全部权限，密码设为 123。注意这里唯独不包括grant的权限。

```
1 GRANT ALL PRIVILEGES ON *.* TO joe@'%' IDENTIFIED BY '123';
```

- ALL PRIVILEGES是表示所有权限，你也可以使用SELECT、UPDATE等权限。
- ON用来指定权限针对哪些库和表。
- . 中前面的 * 号用来指定数据库名，后面的 * 号用来指定表名。这里的 * 表示所有的。
- TO表示将权限赋予某个用户。
- li4@'localhost'表示li4用户，@后面接限制的主机，可以是IP、IP段、域名以及%，%表示任何地方。注意：这里%有的版本不包括本地，以前碰到过给某个用户设置了%允许任何地方登录，但是在本地登录不了，这个和版本有关系，遇到这个问题再加一个localhost的用户就可以了。

- IDENTIFIED BY指定用户的登录密码。
- 如果需要赋予包括GRANT的权限，添加参数“WITH GRANT OPTION”这个选项即可，表示该用户可以将自己拥有的权限授权给别人。经常有人在创建操作用户的时候不指定WITHGRANTOPTION选项导致后来该用户不能使用GRANT命令创建用户或者给其它用户授权。
- 以使用GRANT重复给用户添加权限，**权限叠加**，比如你先给用户添加一个SELECT权限，然后又给用户添加一个INSERT权限，那么该用户就同时拥有了SELECT和INSERT权限。

我们在开发应用的时候，经常会遇到一种需求，就是要根据用户的不同，对数据进行横向和纵向的分组。

- 所谓横向的分组，就是指用户可以接触到的数据的范围，比如可以看到哪些表的数据；
- 所谓纵向的分组，就是指用户对接触到的数据能访问到什么程度，比如能看、能改，甚至是删除。

2.4查看权限

- 查看当前用户权限

```
1 SHOW GRANTS;
2 # 或
3 SHOW GRANTS FOR CURRENT_USER;
4 # 或
5 SHOW GRANTS FOR CURRENT_USER();
```

- 查看某用户的全局权限

```
1 SHOW GRANTS FOR 'user'@'主机地址' ;
```

2.5收回权限

收回权限就是取消已经赋予用户的某些权限。**收回用户不必要的权限可以在一定程度上保证系统的安全性。**MySQL中使用 **REVOKE**语句 取消用户的某些权限。使用REVOKE收回权限之后，用户账户的记录将从db、host、tables_priv和columns_priv表中删除，但是用户账户记录仍然在 user表中保存（删除user表中的账户记录使用DROP USER语句）。

注意：在将用户账户从user表删除之前，应该收回相应用户的所有权限。

- 收回权限命令

```
1 REVOKE 权限1,权限2,...权限n ON 数据库名称.表名称 FROM 用户名@用户地址;
```

- 举例

```
1 #收回全库全表的所有权限
2 REVOKE ALL PRIVILEGES ON *.* FROM joe@'%';
3 #收回mysql库下的所有表的插删改查权限
4 REVOKE SELECT,INSERT,UPDATE,DELETE ON mysql.* FROM joe@localhost;
```

- 注意：须用户重新登录后才能生效

总结

有一些程序员喜欢使用Root超级用户来访问数据库，完全把 权限控制 放在 应用层面 实现。这样当然也是可以的。但建议大家，尽量使用数据库自己的角色和用户机制来控制访问权限，不要輕易用Root账号。因为Root账号密码放在代码里面不安全，一旦泄露，数据库就会完全 失去保护。

而且，MySQL的权限控制功能十分完善，应该尽量利用，可以提高效率，而且安全可靠。

3.权限表

MySQL服务器通过 权限表 来控制用户对数据库的访问，权限表存放在 mysql数据库 中。MySQL数据库系统会根据这些权限表的内容为每个用户赋予相应的权限。这些权限表中最重要的是 user表、 db表。除此之外，还有 table_priv表、 column_priv表和 proc_priv表 等。在MySQL启动时，服务器将这些数据库表中权限信息的内容读入内存。

表名	描述
user	用户账号及权限信息
global_grants	动态全局授权
db	数据库层级的权限
tables_priv	表层级的权限
columns_priv	列层级的权限
procs_priv	存储的过程和函数权限
proxies_priv	代理用户的权限
default_roles	账号连接并认证后默认授予的角色
role_edges	角色子图的边界
password_history	密码更改信息

3.1user表

user表是MySQL中最重要的一个权限表， 记录用户账号和权限信息 ，有49个字段。如下图：

字段名	数据类型	默认值
Host	char(60)	
User	char(16)	
authentication_string	text	
Select_priv	enum('N','Y')	N
Insert_priv	enum('N','Y')	N
Update_priv	enum('N','Y')	N
Delete_priv	enum('N','Y')	N
Create_priv	enum('N','Y')	N
Drop_priv	enum('N','Y')	N
Reload_priv	enum('N','Y')	N
Shutdown_priv	enum('N','Y')	N
Process_priv	enum('N','Y')	N
File_priv	enum('N','Y')	N
Grant_priv	enum('N','Y')	N
References_priv	enum('N','Y')	N
Index_priv	enum('N','Y')	N
Alter_priv	enum('N','Y')	N
Show_db_priv	enum('N','Y')	N
Super_priv	enum('N','Y')	N
Create_tmp_table_priv	enum('N','Y')	N
Lock_tables_priv	enum('N','Y')	N
Execute_priv	enum('N','Y')	N
Repl_slave_priv	enum('N','Y')	N
Repl_client_priv	enum('N','Y')	N
Create_view_priv	enum('N','Y')	N
Show_view_priv	enum('N','Y')	N
Create_routine_priv	enum('N','Y')	N
Alter_routine_priv	enum('N','Y')	N
Create_user_priv	enum('N','Y')	N
Event_priv	enum('N','Y')	N
Trigger_priv	enum('N','Y')	N
Create_tablespace_priv	enum('N','Y')	N
ssl_type	enum('', 'ANY', 'X509', 'SPECIFIED')	
ssl_cipher	blob	NULL
x509_issuer	blob	NULL
x509_subject	blob	NULL
max_questions	int(11) unsigned	0
max_updates	int(11) unsigned	0
max_connections	int(11) unsigned	0
max_user_connections	int(11) unsigned	0
plugin	char(64)	
authentication_string	text	NULL

这些字段可以分成4类，分别是范围列（或用户列）、权限列、安全列和资源控制列。

1.范围列（或用户列）

user表的用户列包括 Host、User、authentication_string，分别表示主机名、用户名和密码。Host指明允许访问的IP或主机范围，User指明允许访问的用户名。其中User和Host为用户表的联合主键。当用户与服务器之间建立连接时，输入的账户信息中的用户名称、主机名和密码必须匹配User表中对应的字段，只有3个值都匹配的时候，才允许连接的建立。这3个字段的值就是创建账户时保存的账户信息。修改用户密码时，实际就是修改user表的authentication_string字段的值。

- host：表示连接类型
 - %表示所有远程通过 TCP方式的连接
 - IP地址 如 (192.168.1.2、127.0.0.1)通过制定ip地址进行的 TCP方式的连接
 - 机器名 通过制定网络中的机器名进行的TCP方式的连接
 - ::1 IPv6的本地ip地址，等同于IPv4的 127.0.0.1
 - localhost本地方式通过命令行方式的连接，比如 mysql-uxxx-pxxx方式的连接。
- user：表示用户名，同一用户通过不同方式链接的权限是不一样的。
- password：密码
 - 所有密码串通过 password(明文字符串)生成的密文字符串。MySQL8.0在用户管理方面增加了角色管理，默认密码加密方式也做了调整，由之前的 SHA1 改为了 SHA2，不可逆。同时加上 MySQL5.7的禁用用户和用户过期的功能，MySQL在用户管理方面的功能和安全性都较之前版本大大的增强了。
 - mysql5.7及之后版本的密码保存到 authentication_string 字段中不再使用password字段。

2.权限列

权限列的字段决定了用户的权限，描述了在全局范围内允许对数据和数据库进行的操作，包括查询权限、修改权限等 普通权限，还包括关闭服务器、超级权限和加载用户等 高级权限。普通权限用于操作数据库；高级权限用于数据库管理。

user表中对应的权限是针对所有用户数据库的。这些字段值的类型为 ENUM，可以取的值只能为Y和N，Y表示该用户有对应的权限；N表示用户没有对应的权限。从user表的结构可以看到，这些字段的值N。如果要修改权限，就可以使用GRANT语句或UPDATE语句更改user表的这些字段来修改用户对应的权限。

- Grant_priv字段
 - 表示是否拥有GRANT权限
- Shutdown_priv字段
 - 表示是否拥有停止MySQL服务的权限
- Super_priv字段
 - 表示是否拥有超级权限
- Execute_priv字段
 - 表示是否拥有EXECUTE权限。拥有EXECUTE权限，可以执行存储过程和函数。
- Select_priv,Insert_priv等
 - 为该用户所拥有的权限。

3.安全列

安全列只有6个字段，其中两个是ssl相关的 (ssl_type、ssl_cipher)，用于 加密；两个是x509相关的 (x509_issuer、x509_subject)，用于 标识用户；另外两个Plugin字段用于 验证用户身份 的插件，该字段不能为空。如果该字段为空，服务器就使用内建授权验证机制验证用户身份。

4.资源控制列

资源控制列的字段用来限制用户使用的资源，包含4个字段，分别为：

1. max_questions，用户每小时允许执行的查询操作次数；
2. max_updates，用户每小时允许执行的更新操作次数；
3. max_connections，用户每小时允许执行的连接操作次数；
4. max_user_connections，用户允许同时建立的连接次数。

一个小时内用户查询或者连接数量超过资源控制限制，用户将被锁定，直到下一个小时才可以再次执行对应的操作，可以使用GRANT语句更新这些字段的值。

查看字段：

```
1 | DESC mysql.user;
```

查看用户,以列的方式显示数据：

```
1 | SELECT * FROM mysql.user \G;
```

查询特定字段：

```
1 | SELECT host,user,authentication_string,select_priv,insert_priv,drop_priv
2 | FROM mysql.user;
```

```
mysql> select host,user,authentication_string,Select_priv from user;
+-----+-----+-----+-----+
| host      | user      | authentication_string          | Select_priv |
+-----+-----+-----+-----+
| localhost | root      | *E56A114692FE0DE073F9A1DD68A00EEB9703F3F1 | Y           |
| localhost | mysql.sys | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | N           |
| %         | zhang3    | *E56A114692FE0DE073F9A1DD68A00EEB9703F3F1 | N           |
| %         | root      | *E56A114692FE0DE073F9A1DD68A00EEB9703F3F1 | Y           |
+-----+-----+-----+-----+
4 rows in set (0.01 sec)
```

3.2db表

db表是MySQL数据中非常重要的权限表，db表中存储了用户对哪个数据库的权限。决定用户能从哪个主机存取个数据库，db表比较常用。

user表中的权限是针对所有数据库的，如果user表中的Select_priv字段取值为Y，那么该用户可以查询所有数据库中的表，**如果希望用户只对某个数据库有操作权限，那么需要将user表中对应的权限设置为N**，然后在db表中设冒对应数据库的操作权限。由此可知，用户先根据user表的内容获取权限，然后根据db表的内容获取权限。

使用DESCRIBE查看db表的基本结构具体SQL语句如下：

```
1 | DESCRIBE mysql.db;
```

字段名	数据类型	默认值
Host	char(60)	
Db	char(64)	
User	char(32)	
Select_priv	enum('N','Y')	N
Insert_priv	enum('N','Y')	N
Update_priv	enum('N','Y')	N
Delete_priv	enum('N','Y')	N
Create_priv	enum('N','Y')	N
Drop_priv	enum('N','Y')	N
Grant_priv	enum('N','Y')	N
References_priv	enum('N','Y')	N
Index_priv	enum('N','Y')	N
Alter_priv	enum('N','Y')	N
Create_tmp_table_priv	enum('N','Y')	N
Lock_tables_priv	enum('N','Y')	N
Create_view_priv	enum('N','Y')	N
Show_view_priv	enum('N','Y')	N
Create_routine_priv	enum('N','Y')	N
Alter_routine_priv	enum('N','Y')	N
Execute_priv	enum('N','Y')	N
Event_priv	enum('N','Y')	N
Trigger_priv	enum('N','Y')	N

执行结果显示，db表的字段大致可以分为两类，分别为用户列和权限列。

1.用户列

db表用户列有3个字段，分别是Host、User、Db。这3个字段分别表示主机名、用户名和数据库名。表示从某个主机连接某个用户对某个数据库的操作权限，这3个字段的组合构成了db表的主键。

2.权限列

Create_routine_priv和Alter_routine_priv这两个字段决定用户是否具有创建和修改存储过程的权限。

3.tables_priv表和 columns_priv表

tables_priv表用来对表设置操作权限，columns_priv表用来对表的某一列设置权限。tables_priv表和columns_priv表的结构分别如图：

```
1 desc mysql.tables_priv;
```

字段名	数据类型	默认值
Host	char(60)	
Db	char(64)	
User	char(16)	
Table_name	char(64)	
Grantor	char(77)	
Timestamp	timestamp	CURRENT_TIMESTAMP
Table_priv	set('Select','Insert','Update','Delete','Create','Drop','Grant','References','Index','Alter','Create View','Show view','Trigger')	
Column_priv	set('Select','Insert','Update','References')	

tables_priv表有8个字段，分别是Host、Db、User、Table_name、Grantor、Timestamp、Table_priv和 Column_priv，各个字段说明如下：

- Host、 Db、 User 和 Table_name 四个字段分别表示主机名、数据库名、用户名和表名。
- Grantor表示修改该记录的用户。
- Timestamp表示修改该记录的时间。
- Table_priv 表示对象的操作权限。包括Select、Insert、Update、Delete、Create、Drop、Grant、References、Index和Alter。
- Column_priv字段表示对表中的列的操作权限，包括Select、Insert、Update和References。

```
1 desc mysql.columns_priv;
```

字段名	数据类型	默认值
Host	char(60)	
Db	char(64)	
User	char(16)	
Table_name	char(64)	
Column_name	char(64)	
Timestamp	timestamp	CURRENT_TIMESTAMP
Column_priv	set('Select','Insert','Update','References')	

columns_priv表只有7个字段，分别是Host、 Db、 User、 Table_name、 Column_name、Timestamp、Column_priv。其中，Column_name用来指定对哪些数据列具有操作权限。

3.4procs_priv表

procs_priv表可以对 存储过程和存储函数设置操作权限，表结构如图：

```
1 desc mysql.procs_priv;
```

字段名	数据类型	默认值
Host	char(60)	
Db	char(64)	
User	char(16)	
Routine_name	char(64)	
Routine_type	enum('FUNCTION','PROCEDURE')	NULL
Grantor	char(77)	
Proc_priv	set('Execute','Alter Routine','Grant')	
Timestamp	timestamp	CURRENT_TIMESTAMP

procs_priv表包含8个字段，分别是Host、 Db、 User、 Routine_name、 Routine_type、 Grantor、Proc_priv和Timestamp，各个字段的说明如下：

- Host、 Db 和 User 字段分别表示主机名、数据库名和用户名。
- Routine_name 表示存储过程或函数的过程。
- Routine_type 表示存储过程或函数的类型。Routine_type字段有两个值，分别是FUNCTION和PROCEDURE：FUNCTION表示这是一个存储函数，PROCEDURE表示这是一个存储过程。
- Grantor是插入或修改该记录的用户。
- Proc_priv表示拥有的权限，包括Execute、AlterRoutine、Grant三种。

- Timestamp表示记录更新时间。

4.访问控制(了解)

正常情况下，并不希望每个用户都可以执行所有的数据库操作。当MySQL允许一个用户执行各种操作时，它将首先核实该用户向MySQL服务器发送的连接请求，然后确认用户的操作请求是否被允许。这个过程称为MySQL中的 **访问控制过程**。MySQL的访问控制分为两个阶段：**连接核实阶段** 和 **请求核实阶段**。

4.1连接核实阶段

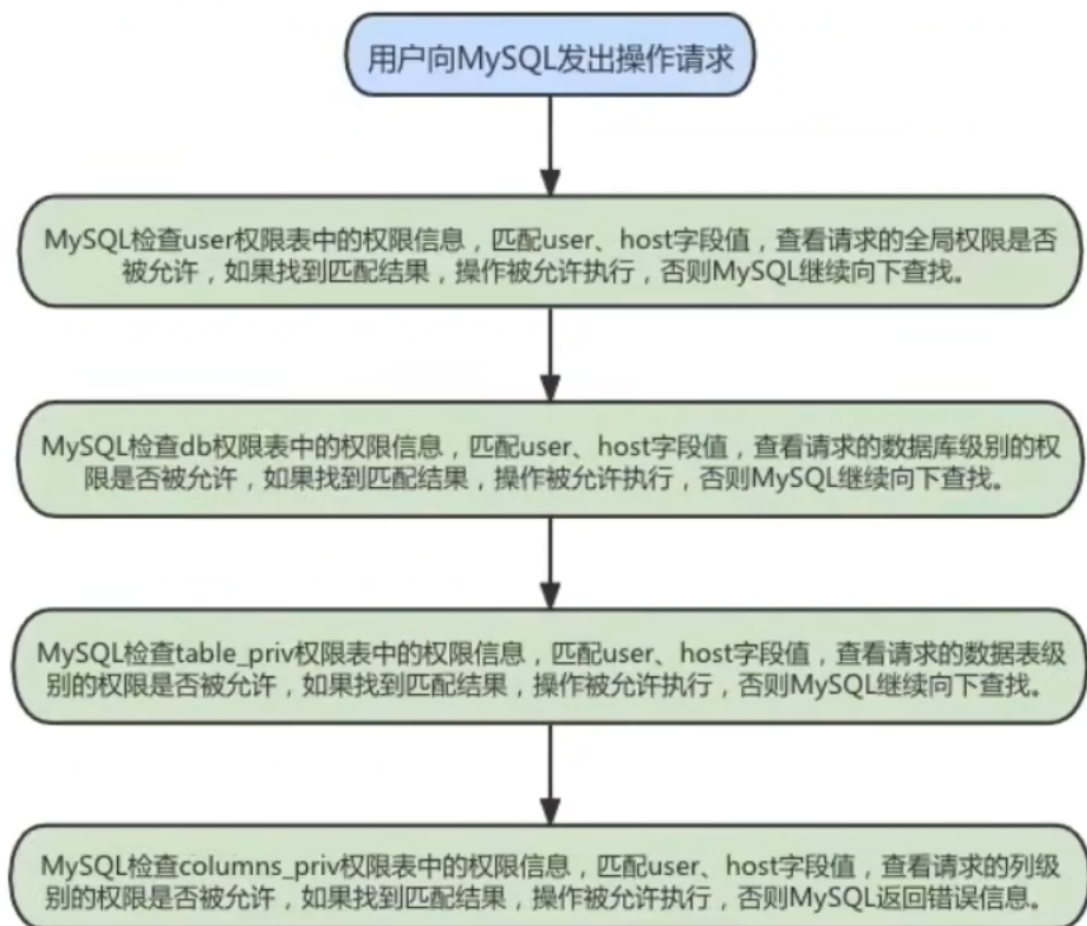
当用户试图连接MySQL服务器时，服务器基于用户的身份以及用户是否能提供正确的密码验证身份来确定接受或者拒绝连接。即客户端用户会在连接请求中提供用户名、主机地址、用户密码，MySQL服务器接收到用户请求后，会使用user表中的host、user和authentication_string这3个字段匹配客户端提供信息。

服务器只有在user表记录的Host和User字段匹配客户端主机名和用户名，并且提供正确的密码时才接受连接。**如果连接核实没有通过，服务器就完全拒绝访问；否则，服务器接受连接，然后进入阶段2等待用户请求。**

4.2请求核实阶段

一旦建立了连接，服务器就进入了访问控制的阶段2，也就是请求核实阶段。对此连接上进来的每个请求，服务器检查该请求要执行什么操作、是否有足够的权限来执行它，这正是需要授权表中的权限列发挥作用的地方。这些权限可以来自user、db、table_priv和column_priv表。

确认权限时，MySQL首先 **检查 user表**，如果指定的权限没有在user表中被授予，那么MySQL就会继续 **检查db表**，db表是下一安全层级，其中的权限限定于数据库层级，在该层级的SELECT权限允许用户查看指定数据库的所有表中的数据；如果在该层级没有找到限定的权限，则MySQL继续 **检查 tables_priv表** 以及 **columns_priv表**，如果所有权限表都检查完毕，但还是没有找到允许的权限操作，MySQL将返回 **错误信息**，用户请求的操作不能执行，操作失败。请求核实的过程如图所示。



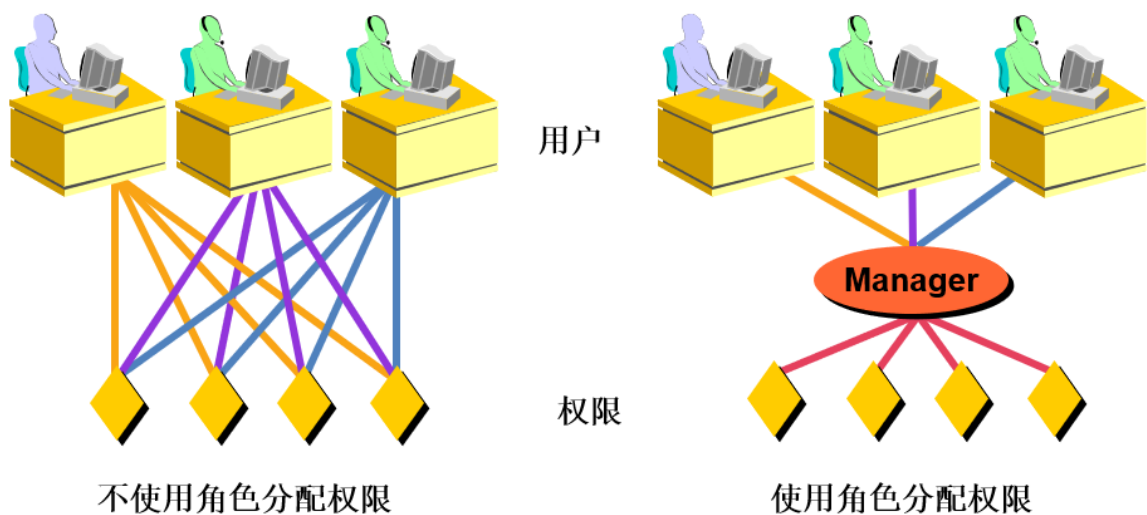
提示：MySQL通过向下层级的顺序（从user表到columns_priv表）检查权限表，但并不是所有的权限都要执行该过程。例如，一个用户登录到MySQL服务器之后只执行对MySQL的管理操作，此时只涉及管理权限，因此MySQL只检查user表。另外，如果请求的权限操作不被允许，MySQL也不会继续检查下一层级的表。

5.角色管理

5.1角色的理解

角色是在MySQL 8.0中引入的新功能。在MySQL中，角色是权限的集合，可以为角色添加或移除权限。用户可以被赋予角色，同时也被授予角色包含的权限。对角色进行操作需要较高的权限。并且像用户账户一样，角色可以拥有授予和撤销的权限。

引入角色的目的是方便管理拥有相同权限的用户。恰当的权限设定，可以确保数据的安全性，这是至关重要的。



5.2创建角色

在实际应用中，为了安全性，需要给用户授予权限。当用户数量较多时，为了避免单独给每一个用户授予多个权限，可以先将权限集合放入角色中，再赋予用户相应的角色。

创建角色使用 `CREATE ROLE` 语句，语法如下：

```
1 CREATE ROLE 'role_name' ['@host_name'] [, 'role_name' ['@host_name']] ...
```

角色名称的命名规则 and 用户名类似。如果 `host_name` 省略，默认为 `%`，`role_name` 不可省略，不可为空。

练习：我们现在需要创建一个经理的角色，就可以用下面的代码：

```
1 CREATE ROLE 'manager' '@localhost';
```

这里创建了一个角色，角色名称是“manager”，角色可以登录的主机是“localhost”，意思是只能从数据库服务器运行的这台计算机登录这个账号。你也可以不写主机名，直接创建角色“manager”：

```
1 CREATE ROLE 'manager';
```

如果不写主机名，MySQL默认是通配符“%”，意思是这个账号可以从任何一台主机上登录数据库。

同样道理，如果我们要创建库管的角色，就可以用下面的代码：

```
1 CREATE ROLE 'stocker';
```

还可以通过如下的指令，一次性创建3个角色：

```
1 CREATE ROLE 'app_developer', 'app_read', 'app_write';
```

5.3给角色赋予权限

创建角色之后，默认这个角色是没有任何权限的，我们需要给角色授权。给角色授权的语法结构是：

```
1 GRANT privileges ON table_name TO 'role_name' ['@host_name'];
```

上述语句中privileges代表权限的名称，多个权限以逗号隔开。可使用 SHOW语句查询权限名称，图11-43列出了部分权限列表。

```
1 | SHOW PRIVILEGES\G;
```

```
mysql> show privileges\G
***** 1. row *****
Privilege: Alter
Context: Tables
Comment: To alter the table
***** 2. row *****
Privilege: Alter routine
Context: Functions,Procedures
Comment: To alter or drop stored functions/procedures
***** 3. row *****
Privilege: Create
Context: Databases,Tables,Indexes
Comment: To create new databases and tables
***** 4. row *****

***** 60. row *****
Privilege: INNODB_REDO_LOG_ENABLE
Context: Server Admin
Comment:
***** 61. row *****
Privilege: INNODB_REDO_LOG_ARCHIVE
Context: Server Admin
Comment:
***** 62. row *****
Privilege: REPLICATION_APPLIER
Context: Server Admin
Comment:
62 rows in set (0.00 sec)
```

练习1：我们现在想给经理角色授予商品信息表、盘点表和应付账款表的只读权限，就可以用下面的代码来实现：

```
1 | GRANT SELECT ON demo.settlement TO 'manager';
2 | GRANT SELECT ON demo.goodsmaster TO 'manager';
3 | GRANT SELECT ON demo.invcount TO 'manager';
```

如果我们需要赋予库管角色盘点表的增删改查权限、商品信息表的只读权限，对应付账款表没有权限，就可以这样：

```
1 | GRANT SELECT,INSERT,DELETE,UPDATE ON demo.invcount TO 'stocker';
2 |
3 | GRANT SELECT ON demo.goodsmaster TO 'stocker';
```

练习2：

```

1 GRANT ALL PRIVILEGES ON app_db.* TO 'app_developer'; -- 给app_db数据库中所有
  表的所有权限
2 GRANT SELECT ON app_db.* TO 'app_read'; -- app_db数据库中所有表的查询权限
3 GRANT INSERT, UPDATE, DELETE ON app_db.* TO 'app_write'; -- app_db数据库中所有表
  的修改权限

```

练习3: 创建三个角色, 分别拥有全部权限、查询权限和读写权限, 步骤如下所示。

(1) 使用如下SQL语句创建三个角色, 角色名为school_admin、school_read、school_write。

```

1 CREATE ROLE 'school_admin', 'school_read', 'school_write';

```

(2) 给每个角色授予对应的权限, school_admin可以对数据库school中的所有表进行任何操作, school_read只能对数据库school中的表进行查询, school_write可以对数据库school中的表进行读写操作, SQL语句如下。

```

1 GRANT ALL PRIVILEGES ON school.* TO 'school_admin';
2 GRANT SELECT ON school.* TO 'school_read';
3 GRANT INSERT, UPDATE, DELETE ON school.* TO 'school_write';

```

5.4查看角色的权限

赋予角色权限之后, 我们可以通过 SHOW GRANTS语句, 来查看权限是否创建成功了:

```

1 mysql> SHOW GRANTS FOR 'manager';
2 +-----+
3 | Grants for manager@% |
4 +-----+
5 | GRANT USAGE ON *.* TO `manager`@`%` |
6 | GRANT SELECT ON `demo`.`goodsmaster` TO `manager`@`%` |
7 | GRANT SELECT ON `demo`.`invcount` TO `manager`@`%` |
8 | GRANT SELECT ON `demo`.`settlement` TO `manager`@`%` |
9 +-----+

```

只要你创建了一个角色, 系统就会自动给你一个“USAGE”权限, 意思是 连接登录数据库的权限。代码的最后三行代表了我们给角色“manager”赋予的权限, 也就是对商品信息表、盘点表和应付账款表的只读权限。

再举例: 来看看库管角色的权限:

```

mysql> SHOW GRANTS FOR 'stocker';
+-----+
| Grants for stocker@% |
+-----+
| GRANT USAGE ON *.* TO `stocker`@`%` |
| GRANT SELECT ON `demo`.`goodsmaster` TO `stocker`@`%` |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `demo`.`invcount` TO `stocker`@`%` |
+-----+

```

结果显示, 库管角色拥有商品信息表的只读权限和盘点表的增删改查权限。

5.5回收角色的权限

角色授权后，可以对角色的权限进行维护，对权限进行添加或撤销。添加权限使用GRANT语句，与角色授权相同。撤销角色或角色权限使用REVOKE语句。

修改了角色的权限，会影响拥有该角色的账户的权限。

撤销角色权限的SQL语法如下：

```
1 REVOKE privileges ON tablename FROM 'rolename';
```

REVOKE privileges ON tablename FROM 'rolename';

练习1：撤销school_write角色的权限。

（1）使用如下语句撤销school_write角色的权限。

```
1 REVOKE INSERT, UPDATE, DELETE ON school.* FROM 'school_write';
```

（2）撤销后使用SHOW语句查看school_write对应的权限，语句如下。

```
1 SHOW GRANTS FOR 'school_write';
```

练习2：

```
1 REVOKE INSERT,UPDATE,DELETE ON app_db.* FROM 'app_write';    #收回角色的权限
```

5.6删除角色

当我们需要对业务重新整合的时候，可能就需要对之前创建的角色进行清理，删除一些不会再使用的角色。删除角色的操作很简单，你只要掌握语法结构就行了。

```
1 DROP ROLE role [,role2]...
```

注意，如果你删除了角色，那么用户也就失去了通过这个角色所获得的所有权限。

练习：执行如下SQL删除角色school_read。

```
1 DROP ROLE 'school_read';
```

5.7给用户赋予角色

角色创建并授权后，要赋给用户并处于激活状态才能发挥作用。给用户添加角色可使用GRANT语句，语法形式如下：

```
1 GRANT role [,role2,...] TO user [,user2,...];
```

在上述语句中，role代表角色，user代表用户。可将多个角色同时赋予多个用户，用逗号隔开即可。

练习：给kangshifu用户添加角色school_read权限。

（1）使用GRANT语句给kangshifu添加school_read权限，SQL语句如下。

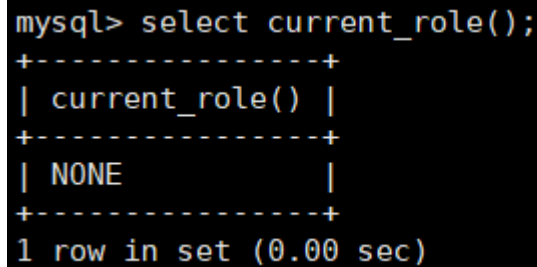

```
1 GRANT 'school_read' TO 'kangshifu'@'localhost';
```

(2) 添加完成后使用SHOW语句查看是否添加成功，SQL语句如下。

```
1 SHOW GRANTS FOR 'kangshifu'@'localhost';
```

(3) 使用kangshifu用户登录，然后查询当前角色，如果角色未激活，结果将显示NONE。SQL语句如下。

```
1 SELECT CURRENT_ROLE();
```



```
mysql> select current_role();
+-----+
| current_role() |
+-----+
| NONE           |
+-----+
1 row in set (0.00 sec)
```

上面结果是NONE，说明用户未具备相应的角色。

或者，你用赋予了角色的用户去登录、操作，你会发现，这个账号没有任何权限。这是因为，MySQL中创建了角色之后，默认都是没有被激活，也就是不能用，必须要手动激活，激活以后用户才能拥有角色对应的权限。

5.8激活角色

激活角色有两种方式：

方式1：使用setdefaultrole命令激活角色

举例：

```
1 SET DEFAULT ROLE ALL TO 'kangshifu'@'localhost';
```

举例：使用 SET DEFAULT ROLE 为下面4个用户默认激活所有已拥有的角色如下：

```
1 SET DEFAULT ROLE ALL TO
2 'dev1'@'localhost',
3 'read_user1'@'localhost',
4 'read_user2'@'localhost',
5 'rw_user1'@'localhost';
```

注意：用户需要退出重新登录，才能看到赋予的角色。

方式2：将activate_all_roles_on_login设置为ON

- 默认情况：

```

1 | mysql> show variables like 'activate_all_roles_on_login';
2 | +-----+-----+
3 | | Variable_name | Value |
4 | +-----+-----+
5 | | activate_all_roles_on_login | OFF |
6 | +-----+-----+
7 | 1 row in set (0.00 sec)

```

- 设置:

```

1 | SET GLOBAL activate_all_roles_on_login=ON;

```

这条 SQL 语句的意思是，对 所有角色永久激活。运行这条语句之后，用户才真正拥有了赋予角色的所有权限。

查看当前会话已激活的角色：

```

1 | SELECT CURRENT_ROLE();

```

5.9 撤销用户的角色

撤销用户角色的 SQL 语法如下：

```

1 | REVOKE role FROM user;

```

练习：撤销 kangshifu 用户的 school_read 角色。

- (1) 撤销的 SQL 语句如下

```

1 | REVOKE 'school_read' FROM 'kangshifu'@'localhost';

```

- (2) 撤销后，执行如下查询语句，查看 kangshifu 用户的角色信息

```

1 | SHOW GRANTS FOR 'kangshifu'@'localhost';

```

执行发现，用户 kangshifu 之前的 school_read 角色已被撤销。

5.10 设置强制角色 (mandatory role)

强制角色是给每个创建账户的默认角色，不需要手动设置。强制角色无法被 REVOKE 或者 DROP。

方式1：服务启动前设置

```

1 | [mysqld]
2 | mandatory_roles='role1,role2@localhost,r3@%.atguigu.com'

```

方式2：运行时设置

```
1 SET PERSIST mandatory_roles = 'role1,role2@localhost,r3@%.example.com'; #系统重
  启后仍然有效
2 SET GLOBAL mandatory_roles = 'role1,role2@localhost,r3@%.example.com'; #系统重
  启后失效
```

5.11小结

MySQL主要管理角色的语句如下：

语句	作用
CREATE ROLE and DROP ROLE	创建和删除角色
GRANT and REVOKE	给角色或者账户分配权限
SHOW GRANTS	显示账户/角色所拥有的权限或者角色
SET DEFAULT ROLE	设置账户默认使用什么角色
SET ROLE	改变当前会话的角色
CURRENT_ROLE () 函数	显示当前会话的角色
mandatory_roles和activate_all_roles_on_login系统变量	允许定义用户登陆时强制的或者激活授权的角色

6.配置文件的使用

6.1配置文件格式

与在命令行中指定启动选项不同的是，配置文件中的启动选项被划分为若干个组，每个组有一个组名，用中括号 [] 扩起来，像这样：

```
1 [server]
2   (具体的启动选项...)
3
4 [mysqld]
5   (具体的启动选项...)
6
7 [mysqld_safe]
8   (具体的启动选项...)
9
10 [client]
11   (具体的启动选项...)
12
13 [mysql]
14   (具体的启动选项...)
15
16 [mysqladmin]
17   (具体的启动选项...)
```

像这个配置文件里就定义了许多个组，组名分别是 `server`、`mysqld`、`mysqld_safe`、`client`、`mysql`、`mysqladmin`。每个组下边可以定义若干个启动选项，我们以`[server]`组为例来看一下填写启动选项的形式（其他组中启动选项的形式是一样的）：

```

1 [server]
2 option1          #这是option1, 该选项不需要选项值
3 option2=value2   #这是option2, 该选项需要选项值
4 ...

```

在配置文件中指定启动选项的语法类似于命令行语法，但是配置文件中指定的启动选项不允加前缀，并且每行只指定一个选项，而且 = 周围可以有空白字符（命令行中选项名、=、选项值之间不允许有空白字符）。另外，在配置文件中，我们可以使用 # 来添加注释，从 # 出现直到行尾的内容都属于注释内容，读取配置文件时会忽略这些注释内容。

6.2启动命令与选项组

配置文件中不同的选项组是给不同的启动命令使用的。不过有两个选项组比较特别：

- [server] 组下边的启动选项将作用于 所有的服务器 程序。
- [client] 组下边的启动选项将作用于 所有的客户端 程序。

下面是启动命令能读取的选项组都有哪些：

启动命令	类别	能读取的组
mysqld	启动服务器	[mysqld]、[server]
mysqld_safe	启动服务器	[mysqld]、[server]、[mysqld_safe]
mysql.server	启动服务器	[mysqld]、[server]、[mysql.server]
mysql	启动客户端	[mysql]、[client]
mysqladmin	启动客户端	[mysqladmin]、[client]
mysqldump	启动客户端	[mysqldump]、[client]

比如，在 /etc/mysql/my.cnf 这个配置文件中添加一些内容：

```

1 [server]
2 skip-networking
3 default-storage-engine=MyISAM

```

然后直接用 `mysqld` 启动服务器程序：

```

1 mysqld

```

虽然在命令行没有添加启动选项，但是在程序启动的时候，就会默认的到我们上边提到的配置文件路径下查找配置文件，其中就包括 /etc/my.cnf。又由于 `mysqld` 命令可以读取 [server] 选项组的内容，所以 `skip-networking` 和 `default-storage-engine=MyISAM` 这两个选项是生效的。你可以把这些启动选项放在 [client] 组里再试试用 `mysqld` 启动服务器程序，就不生效。

6.3特定MySQL版本的专用选项组

我们可以在选项组的名称后加上特定的 MySQL 版本号，比如对于 `[mysqld]` 选项组来说，我们可以定义一个 `[mysqld-5.7]` 的选项组，它的含义和 `[mysqld]` 一样，只不过只有版本号为 5.7 的 `mysqld` 程序才能使用这个选项组中的选项。

6.4同一个配置文件中多个组的优先级

我们说同一个命令可以访问配置文件中的多个组，比如 `mysqld` 可以访问 `[mysqld]`、`[server]` 组，如果在同一个配置文件中，比如 `~/my.cnf`，在这些组里出现了同样的配置项，比如这样：

```
1 [server]
2 default-storage-engine=InnoDB
3
4 [mysqld]
5 default-storage-engine=MyISAM
```

那么，将以最后一个出现的组中的启动选项为准，比方说例子中 `default-storage-engine` 既出现在 `[mysqld]` 组也出现在 `[server]` 组，因为 `[mysqld]` 组在 `[server]` 组后边，就以 `[mysqld]` 组中的配置项为准。

6.5命令行和配置文件中启动选项的区别

在命令行上指定的绝大部分启动选项都可以放到配置文件中，但是有一些选项是专门为命令行设计的，比方说 `defaults-extra-file`、`defaults-file` 这样的选项本身就是为了指定配置文件路径的，再放在配置文件中使用就没啥意义了。

如果同一个启动选项既出现在命令行中，又出现在配置文件中，那么以 **命令行中的启动选项为准**！比如我们在配置文件中写了：

```
1 [server]
2 default-storage-engine=InnoDB
```

而我们的启动命令是：

```
1 mysql.server start --default-storage-engine=MyISAM
```

那最后 `default-storage-engine` 的值就是 `MyISAM`。

7.系统变量（复习）

7.1系统变量简介

MySQL 服务器程序运行过程中会用到许多影响程序行为的变量，它们被称为 MySQL 系统变量。比如：

`max_connections`：允许同时连入的客户端数量

`default_storage_engine`：表的默认存储引擎用系统变量

`query_cache_size`：查询缓存的大小MySQL 服务器程序的系统变量有好几百条，我们就不一一列举了。

MySQL 服务器程序的系统变量有好几百条，我们就不一一列举了。

7.2查看系统变量

查看MySQL服务器程序支持的系统变量以及它们的当前值：

```
1 #查看所有全局变量
2 SHOW GLOBAL VARIABLES;
3 #查看所有会话变量
4 SHOW SESSION VARIABLES;
5 或
6 SHOW VARIABLES;
```

```
1 #查看满足条件的部分系统变量。
2 SHOW GLOBAL VARIABLES LIKE '%标识符%';
3 #查看满足条件的部分会话变量。
4 SHOW SESSION VARIABLES LIKE '%标识符%';
```

由于系统变量实在太多了，所以通常都会带一个LIKE过滤条件来看我们需要的系统变量的值。

```
1 mysql> SHOW VARIABLES LIKE 'default_storage_engine';
2 +-----+-----+
3 | Variable_name | value |
4 +-----+-----+
5 | default_storage_engine | InnoDB |
6 +-----+-----+
7 1 row in set, 1 warning (0.00 sec)
8
9 mysql> SHOW VARIABLES LIKE 'default%';
10 +-----+-----+
11 | Variable_name | value |
12 +-----+-----+
13 | default_authentication_plugin | caching_sha2_password |
14 | default_collation_for_utf8mb4 | utf8mb4_0900_ai_ci |
15 | default_password_lifetime | 0 |
16 | default_storage_engine | InnoDB |
17 | default_table_encryption | OFF |
18 | default_tmp_storage_engine | InnoDB |
19 | default_week_format | 0 |
20 +-----+-----+
21 7 rows in set, 1 warning (0.00 sec)
```

这样就查出了所有以default开头的系统变量的值。

7.3设置系统变量

7.3.1通过启动选项设置

大部分的系统变量都可以通过启动服务器时传送启动选项的方式来进行设置。如何填写启动选项我们总结一下，主要是两种方式：

- 通过命令行添加启动选项

比方说我们在启动服务器程序时用这个命令：

```
1 mysqld --default-storage-engine=MyISAM --max-connections=10
```

- 通过配置文件添加启动选项

我们可以这样填写配置文件：

```
1 [server]
2 default-storage-engine=MyISAM
3 Mmax-connections=10
```

当使用上边两种方式中的任意一种启动服务器程序后，我们再来查看一下系统变量的值：

```
1 mysql> SHOW VARIABLES LIKE 'default_storage_engine';
2 +-----+-----+
3 | Variable_name | value |
4 +-----+-----+
5 | default_storage_engine | MyISAM |
6 +-----+-----+
7 1 row in set, 1 warning (0.00 sec)
8
9 mysql> SHOW VARIABLES LIKE 'max_connections';
10 +-----+-----+
11 | Variable_name | value |
12 +-----+-----+
13 | max_connections | 10 |
14 +-----+-----+
15 1 row in set, 1 warning (0.00 sec)
```

可以看到 `default_storage_engine` 和 `max_connections` 这两个系统变量的值已经被修改了。有一点需要注意的是，对于启动选项来说，如果启动选项名由多个单词组成，各个单词之间用短划线 - 或者下划线 _ 连接起来都可以，但是它对应的系统变量的单词之间必须使用下划线 _ 连接起来。

7.3.2 服务器程序运行过程中设置

对于大部分系统变量来说，它们的值可以在服务器程序运行过程中进行动态修改而无需停止并重启服务器。但是，系统变量有作用范围之分。

设置不同作用范围的系统变量

多个客户端程序可以同时连接到一个服务器程序。对于同一个系统变量，我们有时想让不同的客户端有不同的值。比方说客户端A，他想要当前客户端对应的默认存储引擎为 `InnoDB`，所以他可以把系统变量 `default_storage_engine` 的值设置为 `InnoDB`；客户端B，他想要当前客户端对应的默认存储引擎为 `MyISAM`，所以他可以把系统变量 `default_storage_engine` 的值设置为 `MyISAM`。这样两个客户端拥有不同的默认存储引擎，使用时互不影响，十分方便。但是这样各个客户端都私有一份系统变量会产生这么两个问题：

- 有一些系统变量并不是针对单个客户端的，比如允许同时连接到服务器的客户端数量 `max_connections`，查询缓存的大小 `query_cache_size`，这些公有的系统变量让某个客户端私有显然不合适。
- 一个新连接到服务器的客户端对应的系统变量的值该怎么设置？

为了解决这两个问题，MySQL提出了系统变量作用范围的概念，具体来说作用范围分为这两种：

- `GLOBAL`：全局变量，影响服务器的整体操作。
- `SESSION`：会话变量，影响某个客户端连接的操作。（注：`SESSION` 有个别名叫 `LOCAL`）

在服务器启动时，会将每个全局变量初始化为其默认值（可以通过命令行或选项文件中指定的选项更改这些默认值）。然后服务器还为每个连接的客户端维护一组会话变量，客户端的会话变量在连接时使用相应全局变量的当前值初始化。

以 `default_storage_engine` 举例，在服务器启动时会初始化一个名为 `default_storage_engine`，作用范围为 `GLOBAL` 的系统变量。之后每当有一个客户端连接到该服务器时，服务器都会单独为该客户端分配一个名为 `default_storage_engine`，作用范围为 `SESSION` 的系统变量，该作用范围为 `SESSION` 的系统变量值按照当前作用范围为 `GLOBAL` 的同名系统变量值进行初始化。

很显然，通过启动选项设置的系统变量的作用范围都是 `GLOBAL` 的，也就是对所有客户端都有效的，因为在系统启动的时候还没有客户端程序连接进来呢。了解了系统变量的 `GLOBAL` 和 `SESSION` 作用范围之后，我们再看一下在服务器程序运行期间通过客户端程序设置系统变量的语法：

```
1 SET [GLOBAL | SESSION] 系统变量名=值;
```

或者写成这样也行：

```
1 SET [@@(GLOBAL|SESSION).]var_name = XXX;
```

比如，服务器运行过程中把作用范围为 `GLOBAL` 的系统变量 `default_storage_engine` 的值修改为 `MyISAM`，也就是想让之后新连接到服务器的客户端都用 `MyISAM` 作为默认的存储引擎，那我们可以选择下边两条语句中的任意一条来进行设置：

```
1 # 方式一：
2 SET GLOBAL default_storage_engine = MyISAM;
3 # 方式二：
4 SET @@GLOBAL.default_storage_engine = MyISAM;
```

如果只想对本客户端生效，也可以选择下边三条语句中的任意一条来进行设置：

```
1 # 方式一：
2 SET SESSION default_storage_engine=MyISAM;
3 # 方式二：
4 SET @@SESSION.default_storage_engine = MyISAM;
5 # 方式三：
6 SET default_storage_engine = MyISAM;
```

从上边的方式三可以看出，如果在设置系统变量的语句中省略了作用范围，默认的作用范围就是 `SESSION`。也就是说 `SET 系统变量名 = 值` 和 `SET SESSION 系统变量名 = 值` 是等价的。

查看不同作用范围的系统变量

既然系统变量有作用范围之分，那我们的 `SHOW VARIABLES` 语句查看的是什么作用范围的系统变量呢？

答：默认查看的是 `SESSION` 作用范围的系统变量。

我们也可以在查看系统变量的语句上加上要查看哪个作用范围的系统变量：

```
1 SHOW [GLOBAL|SESSION] VARIABLES [LIKE 匹配的模式];
```

下边我们演示一下完整的设置并查看系统变量的过程：

```
1 mysql> SHOW SESSION VARIABLES LIKE 'default_storage_engine';
```

```

2  +-----+-----+
3  | Variable_name          | value |
4  +-----+-----+
5  | default_storage_engine | InnoDB |
6  +-----+-----+
7  1 row in set, 1 warning (0.00 sec)
8
9  mysql> SHOW GLOBAL VARIABLES LIKE 'default_storage_engine';
10 +-----+-----+
11 | Variable_name          | value |
12 +-----+-----+
13 | default_storage_engine | InnoDB |
14 +-----+-----+
15 1 row in set, 1 warning (0.00 sec)
16
17 mysql> SET SESSION default_storage_engine = MyISAM;
18 Query OK, 0 rows affected (0.00 sec)
19
20 mysql> SHOW SESSION VARIABLES LIKE 'default_storage_engine';
21 +-----+-----+
22 | Variable_name          | value |
23 +-----+-----+
24 | default_storage_engine | MyISAM |
25 +-----+-----+
26 1 row in set, 1 warning (0.00 sec)
27
28 mysql> SHOW GLOBAL VARIABLES LIKE 'default_storage_engine';
29 +-----+-----+
30 | Variable_name          | value |
31 +-----+-----+
32 | default_storage_engine | InnoDB |
33 +-----+-----+
34 1 row in set, 1 warning (0.00 sec)

```

可以看到，最初 `default_storage_engine` 的系统变量无论是在 `GLOBAL` 作用范围上还是在 `SESSION` 作用范围上的值都是 `InnoDB`，我们在 `SESSION` 作用范围把它的值设置为 `MyISAM` 之后，可以看到 `GLOBAL` 作用范围的值并没有改变。

小贴士：如果某个客户端改变了某个系统变量在 `GLOBAL` 作用范围的值，并不会影响该系统变量在当前已经连接的客户端作用范围为 `SESSION` 的值，只会影响后续连入的客户端在作用范围为 `SESSION` 的值

注意事项

并不是所有系统变量都具有 `GLOBAL` 和 `SESSION` 的作用范围。

- 有一些系统变量只具有 `GLOBAL` 作用范围：比方说 `max_connections`，表示服务器程序支持同时最多有多少个客户端程序进行连接。
- 有一些系统变量只具有 `SESSION` 作用范围，比如 `insert_id`，表示在对某个包含 `AUTO_INCREMENT` 列的表进行插入时，该列初始的值。
- 有一些系统变量的值既具有 `GLOBAL` 作用范围，也具有 `SESSION` 作用范围，比如我们前边用到的 `default_storage_engine`，而且其实大部分的系统变量都是这样的
- 有些系统变量是只读的，并不能设置值。

比方说 `version`，表示当前 MySQL 的版本，我们客户端是不能设置它的值的，只能在 `SHOW VARIABLES` 语句里查看。