# MySqlNote

## 表的结构操作

### 删除表

#### 没有外键关联的表

#### 外键关联的表

## 查询（SELECT）

### 匹配查询

#### 模糊（LIKE）

%：代表任意长度的字符串，如b%k可以代表bk,buk,book,break

\_：表示一个字符。如b\_k表示buk，bak，bok

注意：一个汉字是2个字符，因此匹配的时候如果匹配不上（张伟）：

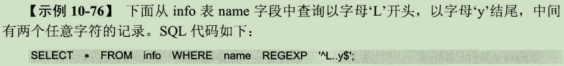
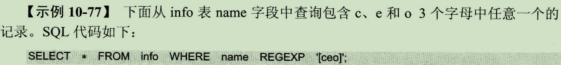
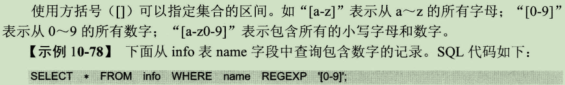
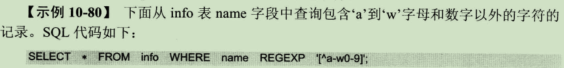
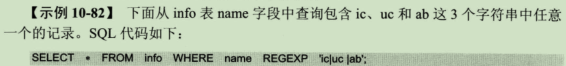
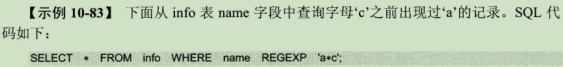
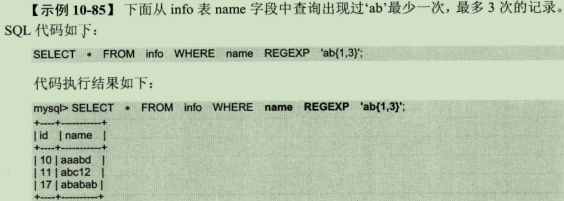
SELECT \* FROM employee WHERE name LIKE ‘张\_’

*可能需要*这样：

SELECT \* FROM employee WHERE name LIKE ‘张\_ \_’

#### 正则表达式（REGEXP）



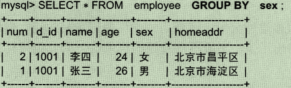
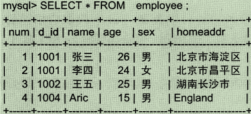
1. ^  
   
2. $  
   
3. .  
   
4. []  
     
   
5. [^ ]  
   
6. |  
   
7. \* +   
   正则表达式中，”\*”和”+”都可以匹配该符号之前的多个字符，但+至少表示一个字符，而\*可以表示0个字符  
   
8. {M,N}表示字符串至少出现M次，最多N次 {M}表示连续出现M次  
   

### 分组（GROUP BY）

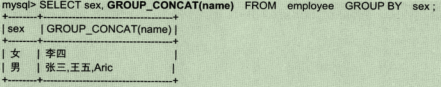
语法：GROUP BY 属性名 [HAVING 条件] [WITH ROLLUP]

“属性名“是指按照该字段的值进行分组，“HAVING 条件 ”是用来限制分组后的结果，“WITH ROLLUP“ 将会在所有记录的最后加上一条记录，该记录是上面所有记录的总和。

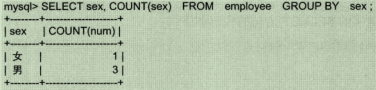
1. 不使用集合函数，则只显示每个分组中的第一条记录，意义不大。



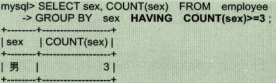
1. **与GROUP\_CONCAT(属性名)**：该函数会将分组中指定的属性，全部显示出来



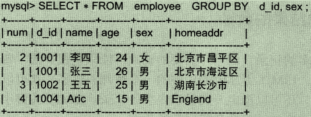
1. **与集合函数**：通常情况下，GROUP BY 与集合函数一起使用，集合函数包括COUNT(),SUM(),AVG(),MAX(),MIN().通常先使用group by 将记录分组，然后对每个分组使用集合函数进行计算。在统计时经常需要使用GROUP BY和集合函数



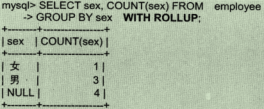
1. **HAVING :**与WHERE一样都是用来限制显示的，区别：where作用与表或者视图，是表和视图的查询条件。HAVING作用于分组后的记录，用于选择满足条件的组。

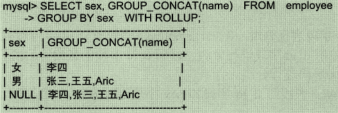


1. **多字段进行分组：**先按照第一个字段分组，如果字段的值相等，则按照第二个字段分组，以此类推。



1. **WITH ROLLUP:** 将在所有记录的最后加上一条记录，该记录是上面所有记录的中使用集合函数列的总和，如：COUNT(),GROUP\_CONCAT()。



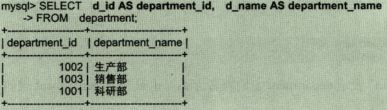


### 别名

**表别名语法：**表名 表的别名



**字段别名语法：** 字段名 [AS] 字段别名。



**注意事项：**

1. 字段的别名只是显示的时候替代字段本来的名称。不像**表别名**可以使用在查询条件中，**字段别名**是不可以在查询条件中使用。
2. 表的别名不能与同数据库中别的表同名，字段的别名不能与同表中其他字段同名。

## 表连接

**外连接与内连接的主要区别**是：内连接仅仅选出两张表中互相匹配的记录，而外连接会选出其他不匹配的记录。

## 索引

### 概述

|  |  |  |  |
| --- | --- | --- | --- |
| 元组 | 属性（字段）1 | 属性（字段）2 | 属性（字段）3 |
| 元组1 |  |  |  |
| 元组2 |  |  |  |

**定义：**

索引是一种数据结构，提高对一个或多个属性的查找速率。就相当于书本里的目录，可以快速找到相关的内容，而不用从头开始一页一页的查找。

索引可以来自任何一个或多个属性，而不必主键。

**作用：**

当关系表变得很大的时候，通过扫描关系表中所有的元组来找出符合查询条件的元组代价太高。

SELECT \*

FROM `city`

WHERE name='Kabul' AND CountryCode='AFG';

如上的SQL语句，如果对CountryCode建立索引，那么就可以快速找到满足‘AFG’的元组，然后再筛选出满足name='Kabul'的元组。

**声明：**

CREATE INDEX nameIndex ON 表名(属性1，属性2……)

对于多属性索引，需要注意的是，最好按照哪些属性可能被查询的次数多少来排序，

如：如果对属性1查询要比属性2要多，则建立（属性1，属性2），反之建立（属性2，属性1）

**优缺点：**

1. 如果在属性上存在索引，则对该属性的进行定值或范围查询（name=’ jizx’或者age<19）,就会极快。
2. 索引的建立会使得对关系的插入，删除和修改变得更加复杂，费时。

**选择：**

代价模型：

一个关系表，被分配到磁盘的多个页面上，每个页面存储多个元组。

查询时，哪怕只检查一个元组也要将包含它的整个磁盘页加载如内存。因此，检查一个页面上所有分组与检查一个元组所花时间几乎没有差别。

**因此减少代价，提高索引效率就是将要加载如内存的页面尽量减少的最少。基于如此，可以参考如下原则建立索引**

1. **主键**建立索引：因为主键上的索引会被频繁的使用，而且键值是唯一的，对于给定的值，最多只有一个元组匹配，最多只有一个磁盘页需要被读入内存
2. **非主键**建立索引：分情况，可能会加速检索速度，可能不会，如下情况的索引是有效的：
   1. 该属性几乎可以看作是一个键，也就是说基本上没有多少元组在该属性上符合给定查询条件的值。因此即使每个满足条件的元组分别在不同磁盘页上，也不需要检索大量的磁盘页。
   2. 元组按该属性在磁盘也上“聚集在一起”。也就是说在该属性上具有相同值的元组，被分组到尽可能少的磁盘页里。在这样的情况下，即使符合条件的元组可能非常多，但是却不必检索与符合要求的元组数目同样多的磁盘页。

例子：

SELECT \*

FROM Movies

WHERE year = 1990；

首先假设Movies的元组不是按照year进行“聚集”的，而是按照title属性的值按照词典排序。那么属性year上的索引基本不会带来任何改进。因为：如果每个磁盘页存储100个元组，那么有相当的几率使得每页至少包含一部1990的电影。这样，用于存储关系Movies的页将会被大量地读入内存。

然而，如果Movies元组是按照year值进行“聚集“的，则利用其上的索引就会找到只有少量的页包含year=1990的元组。

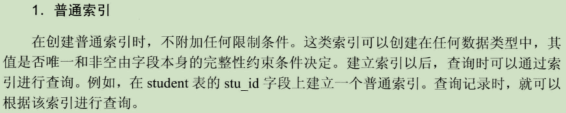
反过来，如果在year和title上都建立索引将不会起任何作用，并且不论是在哪个属性上进行聚集？？？

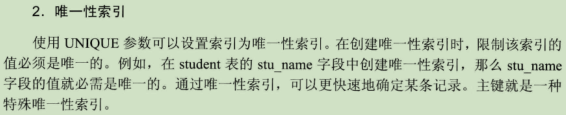
**最佳索引选择：**

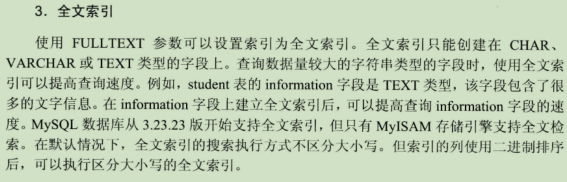
1. 如果插入操作时主要的，只有少量查询，那么就不要索引。
2. 如果某类查询特别频繁，那么就仅仅创建该字段的索引
3. 如果都需要大量的查询，那么可以建立2个或多个索引。

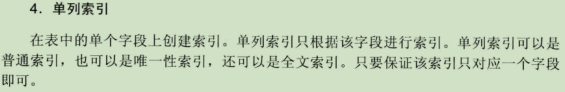
### MySql索引

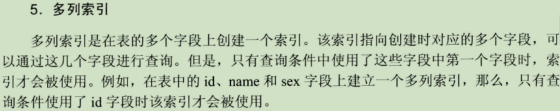
#### 索引类型



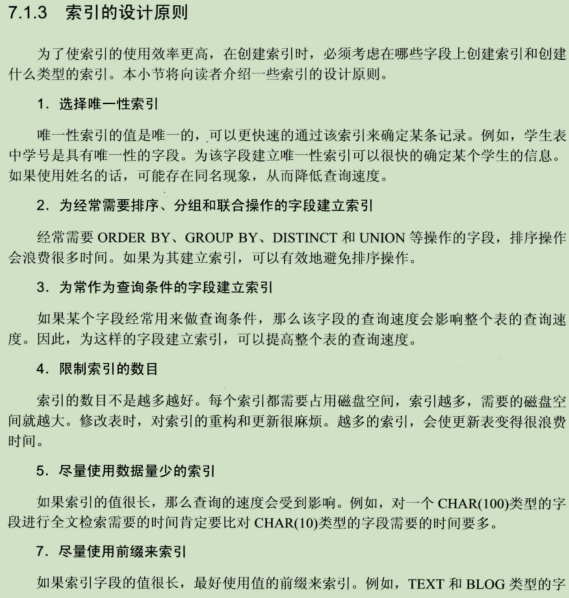








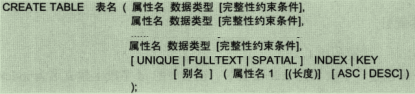
#### 索引创建原则



#### 索引创建与删除

1. **创建表的时候同时创建**

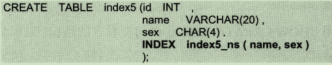
语法：



单列索引：



多列索引：





1. **为已存在的表创建**

语法：



单列索引：



多列索引：



1. **使用ALTER创建**

语法：



单列索引：



多列索引：

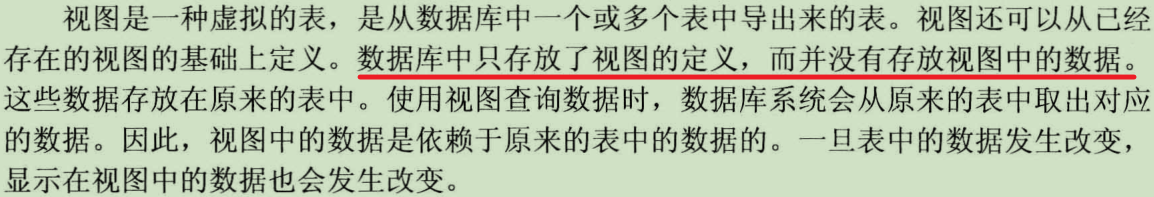


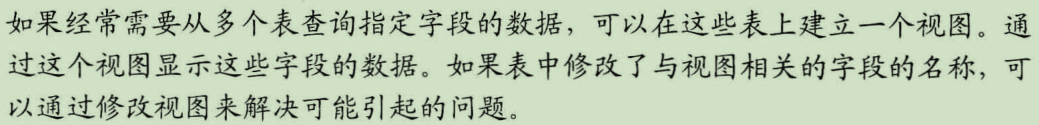
1. **删除索引**



## 视图

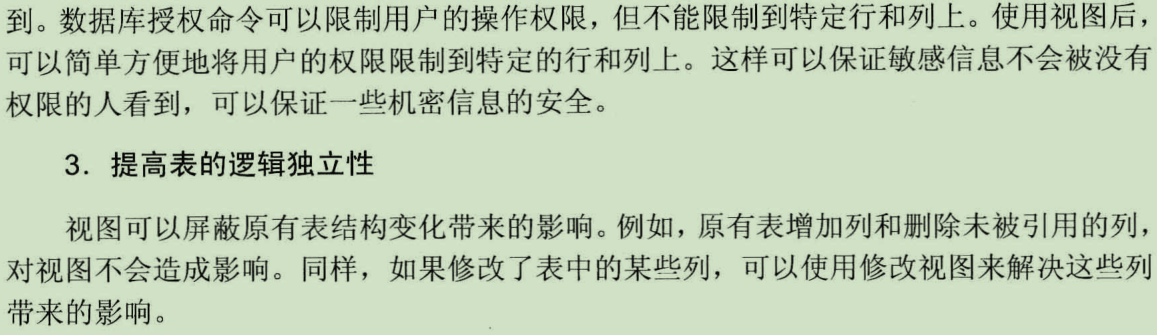
### 视图的定义



相当于子查询，可以将视图替换为对应的定义，即可认为是子查询。

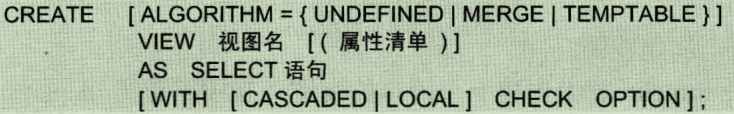
### 视图的意义





### 创建视图

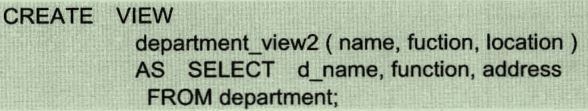
语法：



例子：



自定义属性名

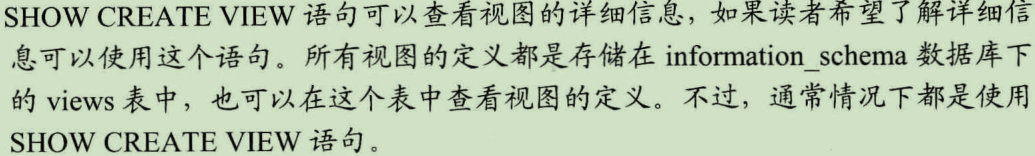


### 查看视图定义

SHWO CREATE VIEW 视图名

或者

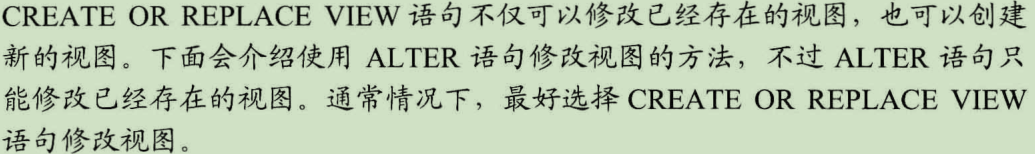
SELECT \* FROM information\_schema.views;



### 修改视图

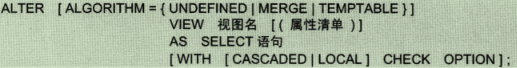
与创建视图的语法类似

#### CREATE OR REPLACE



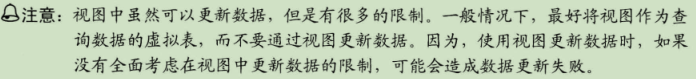


#### ALTER



### 更新视图

#### 最好少更新视图



#### 更新的限制

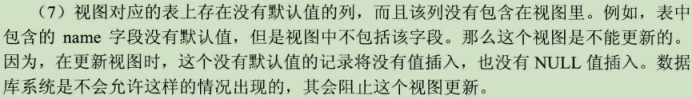












### 删除视图

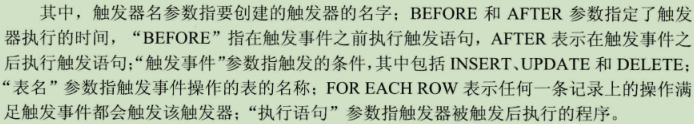
可以同时删除多个视图



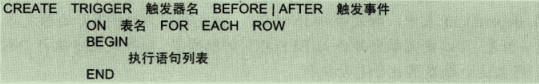
## 触发器

### 创建单执行语句的触发器

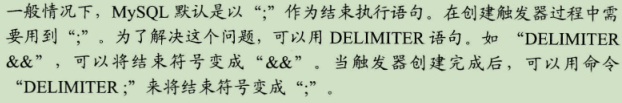


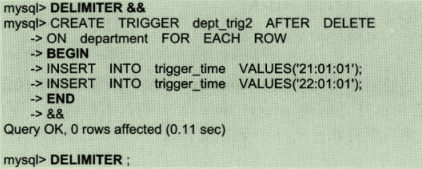


### 创建多执行语句的触发器



分号的冲突：分号表示SQL语句结束符，与多条语句触发器定义冲突，因此需要用DELIMITER定义新的接受符。具体例子如下：





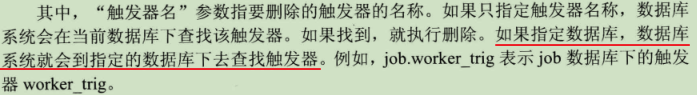
### 查看触发器定义

1. 方法1：SHOW TRIGGERS; 局限性：只能看全部触发器，无法筛选
2. 方法2：查看在information\_schema数据库下的triggers表中存放的数据



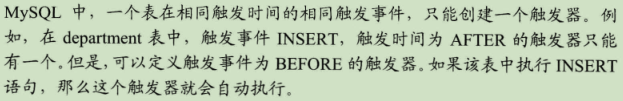
### 删除触发器





### 限制

在MySQL中，一张表在相同触发时间的相同触发事件（INSERT、UPDATE、DELETE）只能创建一个触发器



## 函数

### 数字

#### 四舍五入

FORMAT(x,n) 等同于ROUND(x,n)，都是对小数x保留n位

### 字符串

### 加密

#### PASSSWORD(str)

函数一般是用来给用户的密码加密

#### MD5(str)

主要对普通数据进行加密

#### ENCODE(str，pswd\_str)

使用pswd\_str加密str，加密的结果是一个二进制数，必需用BLOB类型的字段来保存。

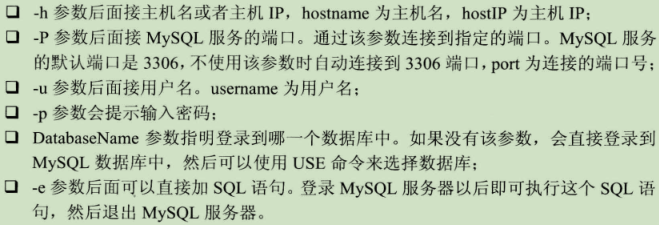
#### DECODE(crypt\_str，pswd\_str)

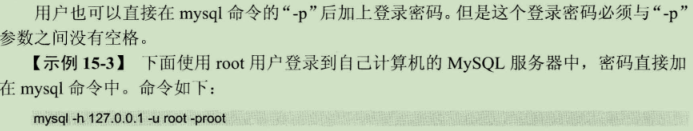
用pswd\_str给crypt\_str解密，其中crypt\_str是经过ENCODE加密的二进制数据

## 用户管理

#### 登录服务器

Mysql –h hostname|hostIP –P port –u username –p DatabaseName –e “SQL语句”



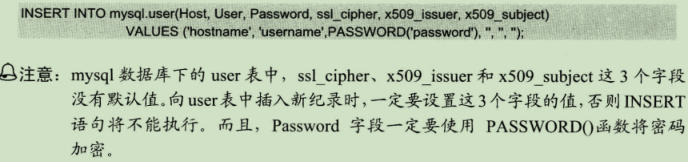


#### 创建用户

**User由username@host组成**

##### INSERT (可能不好使)

INSERT INTO mysql.user（Host,User,Password，ssl\_cipher,x509\_issuer,x509\_subject） VALUE（’hostname’,’username’,PASSWORD(’password’),’’,’’,’’）



执行完后刷新：FLUSH PRIVILEGES;(需要有reload权限)

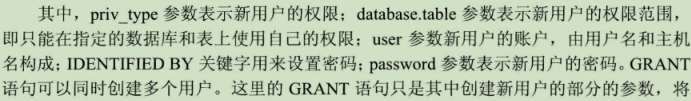
##### CREATE USER

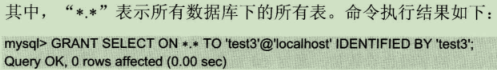
CREATE USER ‘user’ [IDENTIFIED BY ‘password’] [,‘user’ [IDENTIFIED BY ‘password’] ]



##### GRANT(最强大)

GRANT priv\_type ON database.table TO ‘user’ [IDENTIFIED BY ‘password’] [,‘user’ [IDENTIFIED BY ‘password’] ]





#### 删除用户

##### DROP USER

DROP USER user [,user] 可以同时删除多个用户

##### DELETE

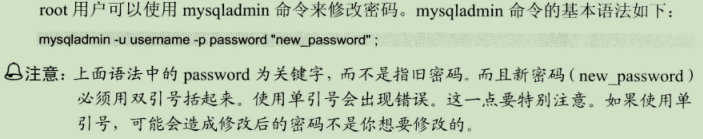
DELETE FROM mysql.user WHERE Host=’hostname’ AND User=’username’;



#### 修改密码

##### root修改root

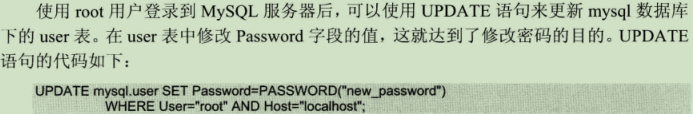
方法1（未登录）：



需要在未登录状态，已登录是执行不了的，如下：



方法2（需登录）可能不好使：



方法3（需登录）：



##### root修改普通用户

方法1（需登录）：

SET PASSWORD FOR ‘username’@’hostname’=PASSWORD(“new\_password”);

方法2（需登录）可能不好使：



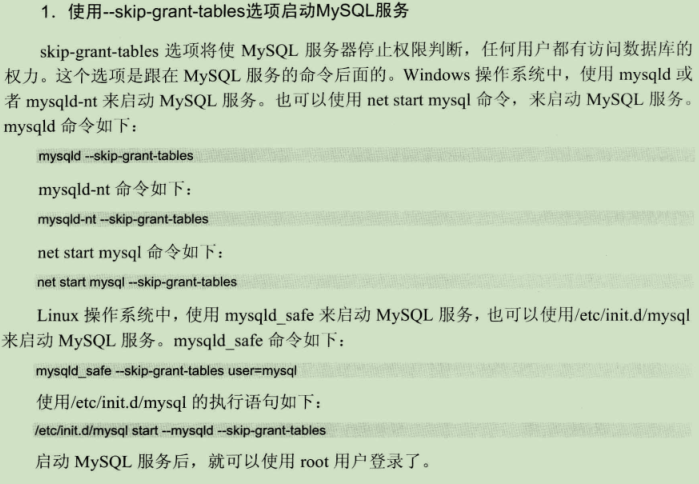
方法3（需登录）：

GRANT priv\_type ON database.table TO user IDENTIFIED BY ‘password’

##### 普通用户修改密码

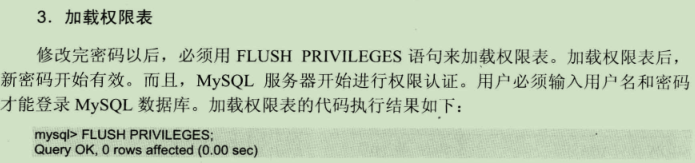
SET PASSWORD=PASSWORD(“new\_password”);

##### root找回密码



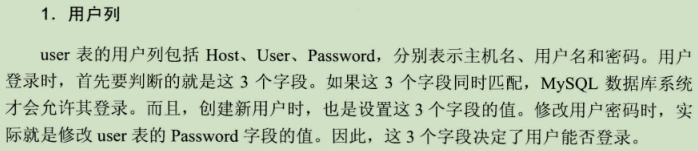
**Mysql5.7不适合以上的安全启动服务方法**

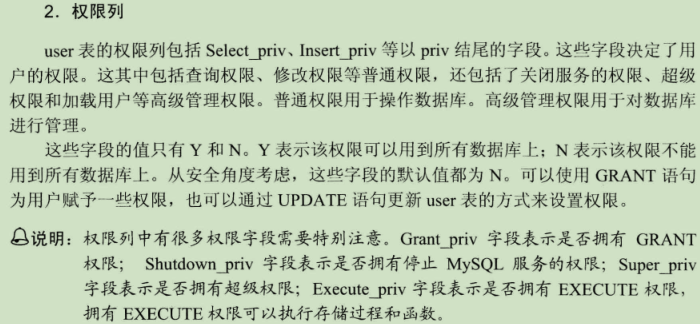


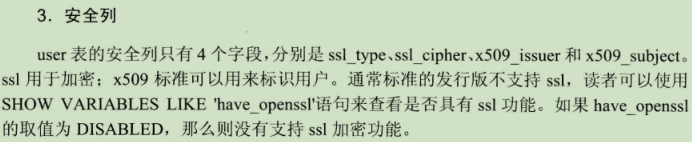


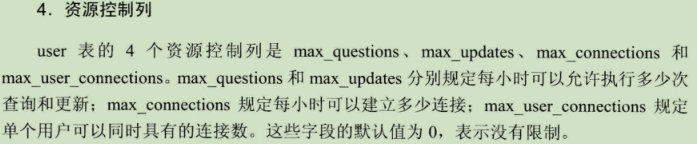
#### 用户权限记录表

Mysql数据库下的user表，db表，host表，table\_priv表，columns\_priv表，procs\_priv表等存储着用户的权限，其中user表是最详细的权限表，其中的字段有如下四大模块：

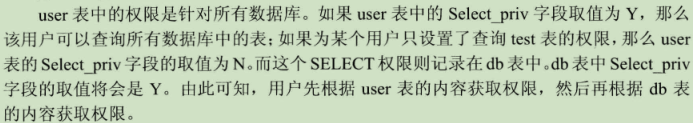


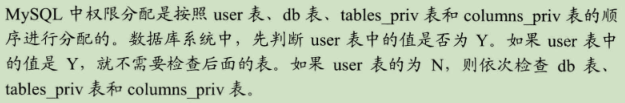






#### 权限查询顺序





#### Grant（授权）

GRANT priv\_type [column\_list]

ON database.table

TO ‘user’ [IDENTIFIED BY ‘password’] [,‘user’ [IDENTIFIED BY ‘password’] ]

[WITH with\_option [with\_option]…]

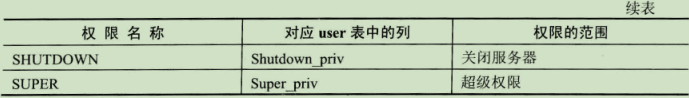
User参数由’username’@’hostname’组成。

With 关键字后面可以带多个参数,分别是:

|  |  |
| --- | --- |
| **CRANT OPTION** | 被授予的用户可以将这些权限再赋予给别的用户 |
| **MAX\_QUERIES\_PER\_HOUR count** | 设置每小时可以允许执行的查询次数count |
| **MAX\_UPDATES\_PER\_HOUR count** | 设置每小时可以允许执行的更新次数count |
| **MAX\_CONNECTIONS\_PER\_HOUR count** | 设置每小时可以允许建立的连接次数count |
| **MAX\_USER\_CONNECTIONS count** | 设置单个用户可以同时具有的连接数 |

##### 权限表





##### 授予全部权限：

GRANT ALL PRIVILEGES ON \*.\* TO user WITH GRANT OPTION

#### Revoke（撤权）

REVOKE priv\_type [column\_list]

ON database.table

FROM ‘user’ [,user]

##### 收回全部权限：

REVOKE ALL PRIVILEGES,GRANT OPTION FROM user；

#### 查看用户权限（SHOW GRANTS）

方法1（麻烦）：

Select \* from mysql.user where user=’’ AND host=’’

方法2：

SHOW GRANTS FOR ‘username’@’hostname’