**/\*\*\***

**Mysql 基础 – 使用Mysql 5.5 版本 – sqlyog可视化工具 – 2018.03**

**注：符号 – 之后为在Mysql终端下的命令,且如果有输出，会在相应的命令行下贴出截图：**



**注：命令包含符号 ； ，且后面的括号为我个人的注解；**

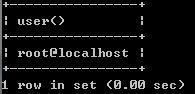
**\*\*/**

**一：Mysql用户与密码相关操作：(Mysql密码是动态md5加密，是不可逆向查询的)**

**1，查看**

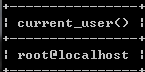
**1.1，查看当前登陆状态的用户名：**

a)- select user();



b)- select current\_user();

可以不加括号；



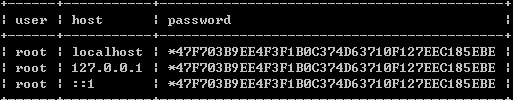
**(注：Mysql大小写不敏感)**

**1.2，查询Mysql中所有的用户名：**

首先使用root账户登录进Mysql终端；在Mysql中其实有一个内置且名为mysql的数据库，这个数据库中存储的是Mysql的一些数据，比如用户、权限信息、存储过程等。

所以按照下面步骤完成查询：

* use mysql;
* select user, host, password from user;



(或者使用命令：select user, host, password from mysql.user; 直接达到上面两条命令的效果)

**1.3，查看用户权限**：

- show grants for root@localhost; (root@localhost为用户名)



**2，修改：**

**2.1，创建新的用户，删除用户：**

a)- create user 'user\_name'@'host' [identified by 'password']；(创建,[]内为选填部分)

b)- insert into mysql.user (host, user, password) values； ('%','john', password('123')); (直接向user表中插入用户)

- flush privileges; (完成创建后最好进行刷新)

user\_name：要创建用户的名字；host：表示要这个新创建的用户允许从哪台机登陆，如果只允许从本机登陆，则填‘localhost’，如果允许从远程登陆，则填 ‘%’；password：新创建用户的登陆数据库密码，如果没密码可以不写；

* drop user ‘user\_name’@‘host’; (删除)

**2.2，设置和更改用户密码：**

- set password for ‘username’@‘host’ = password(‘newpassword’)；

或者直接给当前用户设置密码：

* set password = password('newpassword')；

**2.3，授予用户权限：**

- grant privileges on databasename.tablename to ‘username’@‘host’；

privileges：表示要授予什么权力，多个权限可以用逗号隔开，例如可以有 select ， insert ，delete，update等，如果要授予全部权力，则填 ALL；databasename.tablename：表示用户的权限能用在哪个库的哪个表中，如果想要用户的权限很作用于所有的数据库所有的表，则填 \*.\*，\*是一个通配符，表示全部；‘username’@‘host’：表示授权给哪个用户。

**注：当数据库存在相应的用户的时候GRANT会对用户进行授权，但当数据库不存在该用户的时候，就会创建相应的用户并进行授权；还可以在最后加上 [identified by 'password']**

**2.4，删除用户权限：**

- revoke privileges on database.tablename **from** ‘username’@‘host’；

**2.5，修改远程访问权限：**

a) - update user set host = ‘%’ where user = ‘root’;

将mysql.user 表中host字段的值改为%就表示在任何客户端机器上能以root用户登录到mysql服务器，建议在开发时设为%

b)直接创建用户的时候把localhost设置为%。(%为通配符)。

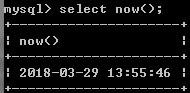
**二：mysql中的数据类型：**

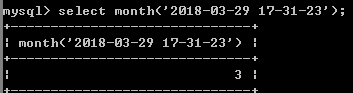
主要有三种，分别为：数值类型，日期类型，以及字符串类型；

**1，Mysql 中的日期：**

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 字节 | 格式 | 用途 |
| DATE | 3 | yyyy-MM-dd | 存储日期值 |
| TIME | 3 | HH:mm:ss | 存储时分秒 |
| YEAR | 1 | yyyy | 存储年 |
| DATETIME | 8 | yyyy-MM-dd HH:mm:ss | 存储日期+时间 |
| TIMESTAMP | 4 | yyyy-MM-dd HH:mm:ss | 存储日期+时间，可作时间戳 |

* select now（）；（获取当前的日期与时间）



* select curdate();（获取当前日期）
* select curtime();（获取当前时间）
* select date('2017-05-28 17:22:14')；（获取日期）
* select month（’2018-03-29 15:23:34’）；(获取月份)
* 
* 依次有day，second，minute，hour，year，time等。

**2，Mysql中的字符串类型：**

1，char：固定长度字符串，如 char(10)必须十位；

2，varchar：可变长度字符串， 如varchar(10)最多十位；

3，text：大文本；

4，enum：枚举类型，每次只能取一个元素；

5，set： 集合类型(能取多个元素)；

|  |  |  |
| --- | --- | --- |
| 类型 | 字节 | 用途 |
| CHAR | 0-255字节 | 定长字符串 |
| VARCHAR | 0-65535 字节 | 变长字符串 |
| TINYBLOB | 0-255字节 | 不超过 255 个字符的二进制字符串 |
| TINYTEXT | 0-255字节 | 短文本字符串 |
| BLOB | 0-65 535字节 | 二进制形式的长文本数据 |
| TEXT | 0-65 535字节 | 长文本数据 |
| MEDIUMBLOB | 0-16 777 215字节 | 二进制形式的中等长度文本数据 |
| MEDIUMTEXT | 0-16 777 215字节 | 中等长度文本数据 |
| LONGBLOB | 0-4 294 967 295字节 | 二进制形式的极大文本数据 |
| LONGTEXT | 0-4 294 967 295字节 | 极大文本数据 |

**2.1 CHAER、VARCHAR:**

char的长度**固定**为创建表时声明的长度。长度可以为从0到255的任何值。当保存char值时，在它们的右边填充空格以达到指定的长度。当检索到char值时，尾部的空格被删除掉，所以，我们在存储时字符串**右边不能有空格**，即使有，查询出来后也会被删除。在存储或检索过程中不进行大小写转换。所以当char类型的字段为唯一值时，添加的值是否已经存在以不包含末尾空格（可能有多个空格）的值确定，比较时会在末尾补满空格后与现已存在的值比较。

varchar列中的值为可变长字符串。长度可以指定为0到65535之间的值（实际可指定的最大长度与编码和其他字段有关，比如编码格式为utf8时，varchar字段时最大值仅为21844）。同char对比，varchar值保存时只保存需要的字符数，**另加一个字节**来记录长度(**如果列声明的长度超过255，则使用两个字节**)。varchar值保存时不进行填充。当值保存和检索时尾部的空格仍保留，符合标准sql。

严格模式下，如果分配给char或varchar列的值超过列的最大长度，则对值进行裁剪以使其适合，如果裁剪非空格字符，则会出现错误（非严格模式下为警告）。。

因为 varchar 类型可以根据实际内容动态改变存储值的长度，所以在不能确定字段需要多少字符时使用 varchar 类型可以大大地节约磁盘空间、提高存储效率。

**总结如下：**

* char是固定长度字符串，其长度范围为0-255且与编码方式无关，无论字符实际长度是多少，都会按照指定长度存储，不够的用空格补足；varchar为可变长度字符串，在utf8编码的数据库中其长度范围为0-21844：
* char实际占用的字节数即存储的字符所占用的字节数，varchar实际占用的字节数为存储的字符+1或+2或+3；
* mysql处理char类型数据时会将结尾的所有空格处理掉而varchar类型数据则不会；
* 4.0版本以下，char/varchar(20)，指的是20字节，如果存放utf8汉字时，只能存6个（每个汉字3字节） ；5.0版本以上，char/varchar(20)，指的是20字符，无论存放的是数字、字母还是utf8汉字（每个汉字3字节），都可以存放20个；

**2.2 BINARY、VARBINARY:**

binary和varbinary类型类似于char和varchar类型，但是不同的是，它们存储的不是字符字符串，而是二进制串。

当保存binary值时，在它们右边填充0x00(零字节)值以达到指定长度。取值时不删除尾部的字节。对于varbinary，插入时不填充字符，选择时不裁剪字节。

**2.3 BLOB、TEXT:**

blob是一个二进制大对象，可以容纳可变数量的数据。有4种blob类型：tinyblob、blob、mediumblob和longblob。它们只是可容纳值的最大长度不同。

有4种text类型：tinytext、text、mediumtext和longtext。这些对应4种blob类型，有相同的最大长度和存储需求。

blob 可以储存图片，text不行，text只能储存纯文本文件。在blob和text类型之间的唯一差别是对blob值的排序和比较以大小写敏感方式执行，而对 text值是大小写不敏感的。如果要储存中文则选择text

未运行在严格模式时，如果你为blob或text列分配一个超过该列类型的最大长度的值，值被截取以保证适合。如果截掉的字符不是空格，将会产生一条警告。使用严格SQL模式，会产生错误，并且值将被拒绝而不是截取并给出警告。**Mysql单行最大数据量为64K。**

* varchar只能用63352/65533个字节，但是text可以65535个字节全部用起来；
* text不允许有默认值，varchar允许有默认值；

varchar和text两种数据类型，使用建议是能用varchar就用varchar而不用text（存储效率高），varchar(M)的M有长度限制，之前说过，如果大于限制，可以使用mediumtext（16M）或者longtext（4G）。

注：

Strict Mode下有以下限制：

1).不支持对not null字段插入null值

2).不支持对自增长字段插入”值，可插入null值

3).不支持 text 字段有默认值

**3， Mysql数值类型：**

MySQL支持所有标准SQL数值数据类型。这些类型包括严格数值数据类型(integer、smallint、decimal和numeric)，以及近似数值数据类型(float、real和double precision)。

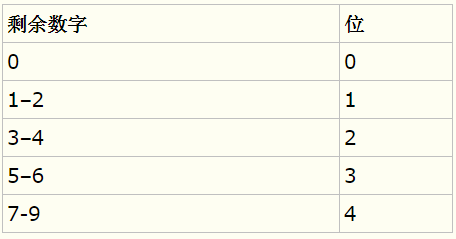
关键字int是integer的同义词，关键字dec是decimal的同义词。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 字节 | 范围（有符号 | 范围（无符号 | 用途 |
| TINYINT | 1 字节 | (-128，127) | (0，255) | 小整数值 |
| SMALLINT | 2 字节 | (-32 768，32 767) | (0，65 535) | 大整数值 |
| MEDIUMINT | 3 字节 | (-8 388 608，8 388 607) | (0，16 777 215) | 大整数值 |
| INT或INTEGER | 4 字节 |  | (0，4 294 967 295) | 大整数值 |
| BIGINT | 8 字节 | 不写了 |  | 极大整数值 |
| FLOAT | 4 字节 | 和上面类似 | 字节思考一下 | 单精度 浮点数值 |
| DOUBLE | 8 字节 |  | 太大了，不写了 | 双精度 浮点数值 |
| DECIMAL | MAX（M,D）+2 | 依赖于M和D的值 | 依赖于M和D的值 | 小数值 |

**3.1 DECIMAL、FLOAT、DOUBLE：**

p是表示有效数字数的精度。 p范围为1-65；d是表示小数点后的位数；d的范围是0-30，mysql要求d小于或等于(<=)p。decimal(p，d)表示列可以存储d位小数的p位数。

mysql分别为整数和小数部分分配存储空间。 mysql使用二进制格式存储decimal值。它将9位数字包装成4个字节。对于每个部分，需要4个字节来存储9位数的每个倍数。剩余数字所需的存储如下表所示：例如，decimal(19,9)对于小数部分具有9位数字，对于整数部分具有19位= 10位数字，小数部分需要4个字节。 整数部分对于前9位数字需要4个字节，1个剩余字节需要1个字节。decimal(19,9)列总共需要9个字节。系统默认为decimal（10,0）。



mysql 支持的三个浮点类型是 float、double 和 decimal 类型。float 数值类型用于表示单精度浮点数值，而 double 数值类型用于表示双精度浮点数值。与整数一样，这些类型也带有附加参数：一个显示宽度指示器和一个小数点指示器。比如语句 float(7,3) 规定显示的值不会超过 7 位数字，小数点后面带有 3 位数字。对于小数点后面的位数超过允许范围的值，mysql 会自动将它**四舍五入为最接近它的值**，再插入它。

unsigned 和 zerofill 修饰符也可以被 float、double 和 decimal 数据类型使用。并且效果与 int 数据类型相同

**注：** float占4个字节，double占8个字节，decimail(m,d)占m+2个字节。float、double类型存在精度丢失问题，即写入数据库的数据未必是插入数据库的数据，而decimal无论写入数据中的数据是多少，都不会存在精度丢失问题，这就是我们要引入decimal类型的原因，decimal类型常见于银行系统、互联网金融系统等对小数点后的数字比较敏感的系统中。

double 和 float 的区别是double精度高，有效数字16位（float精度7位）。但double消耗内存是float的两倍，占8字节，double的运算速度比float慢得多。

**3.2 INT：**

unsigned 修饰符规定字段只保存正值。因为不需要保存数字的正、负符号，可以在储时节约一个“位”的空间。从而增大这个字段可以存储的值的范围。

**三：数据库基础操作：（作为例子的各数据库，表并不是统一的）**

**创建数据库：两种方法：**

* create database my\_db;



* create database if not exists my\_db;



**删除数据库：两种方法**：

* drop database my\_db;



* drop database if exists my\_db;



**创建表：**

* create table table\_name (column\_name column\_type); （标准语法）
* create table tb\_test( **（也可以使用create table if not exists tb\_test( 语句**)

id INT NOT NULL AUTO\_INCREMENT,

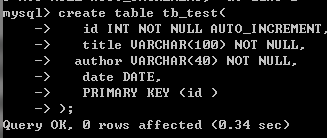
title VARCHAR(100) NOT NULL,

author VARCHAR(40) NOT NULL,

date DATE,

PRIMARY KEY (id )

); (此为例句)



备注：字段使用NOT NULL属性，是因为我们不希望这个字段的值为NULL。 因此，如果用户将尝试创建具有NULL值的记录，那么MySQL会产生错误；字段的AUTO\_INCREMENT属性告诉MySQL自动增加id字段下一个可用编号；关键字PRIMARY KEY用于定义此列作为主键。可以使用逗号分隔多个列来定义主键；auto\_increment只是MySQL特有的

* create table tb\_test(

id INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

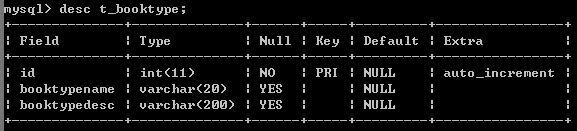
title VARCHAR(100) NOT NULL,

author VARCHAR(40) NOT NULL,

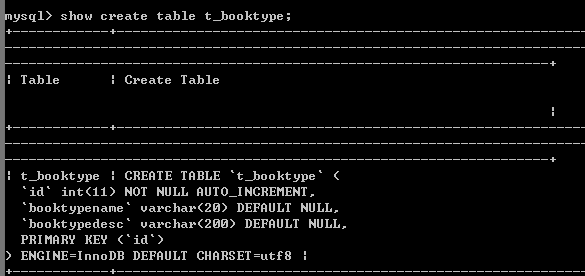
date DATE）；（上面的例句也可以这样写）

**查看表结构**：

* desc + 表名：（第一种方法，显示表结构，字段类型，主键，是否为空等属性，但不显示外键）



* show create table + 表名：（第二种方法，查看表生成的DDL）



**修改表：**

* alter table 旧的表名 rename 新表名；（修改表名）



* alter table 表名 change 原有字段 新的字段 varchar(20); （修改字段）

（把表中原有字段修改为新的字段）



* alter table 表名 add 字段名 int after author; （添加字段）

（在author之后添加新的字段，int为字段类型，和上面例子的表并不是同一个）



* alter table 表名 add 字段名 int first;(添加在第一行)



* alter table 表名 drop 字段名; （删除字段）



5，删除表：

* drop table +表名；（删除表）



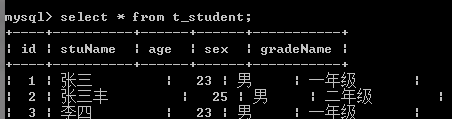


**查询数据：**

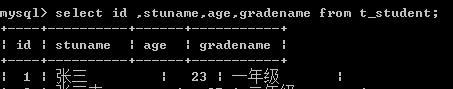
**第一种：单表查询：**

查询所有字段：

* select \* from 表名; （查询所有字段的所有内容）

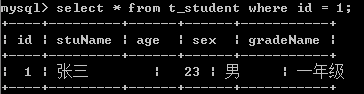


* select id, stuname, age, gradename from t\_student; (查询指定字段)

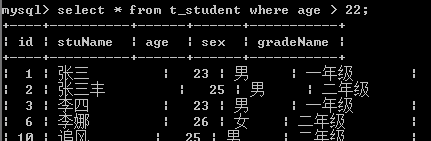


Where 条件查询：

* select \* from 表名 where id = 1; （查询id=1对应的所有字段）

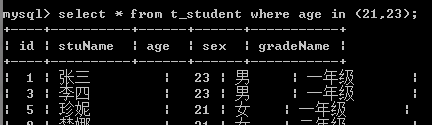


* select \* from 表名 where age > 22; （查询满足条件的所有字段）



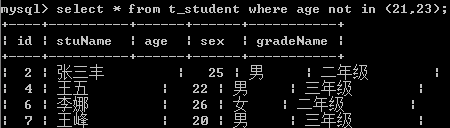
带in关键字查询：

* select \* from 表名 where age in (21,23); （查询满足条件的字段）



带 not in 关键字查询：

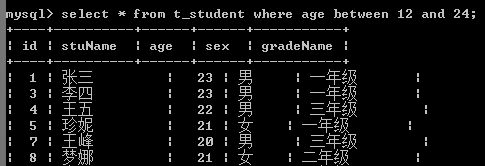
* select \* from t\_student where age not in (21,23); （查询满足条件的所有字段）



带between and 关键字查询：范围查询

* select \* from 表名 where age between 12 and 24;

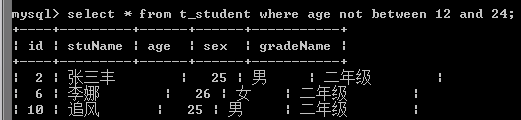
（查询age在12-24之间的所有字段）



带 not between and 关键字查询：不在范围之内

* select \* from t\_student where age not between 12 and 24;

（查询age不在12-24之间的所有字段）



**带like的模糊查询：%代表任意字符，\_代表单个字符；**

* select \* from 表名 where stuname like ‘张三’;

（查询stuname为张三的所有字段）

* Select \* from t\_student where stuname like ‘张三\_’;

（查询stuname为张三\_的所有字段,如张三丰，张三加）

* Select \* from t\_student where stuname like ‘%张三%’;

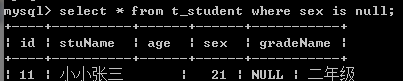
（查询stuname为带有张三的所有字段）

注：%为通配符，\_只匹配任意一个字符；**上下文中的这些\*（所有字段）查询，也可以设置为指定字段查询；**

控制查询：

* select \* from t\_student where sex is null;

（sex列未填写任何内容的行对应的所有字段）

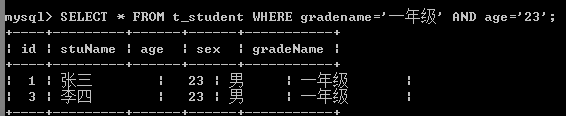


* select \* from t\_student where sex is not null;

And条件查询：

* select \* from t\_student where gradename='一年级' and age='23';

（同时满足这两个条件的行可以输出指定或者所有的字段）



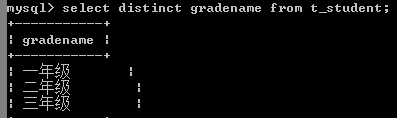
Or条件查询：

- select \* from t\_student where gradename='一年级' or age='23';

（满足这两个条件任意一个的行可以输出指定或者所有的字段）

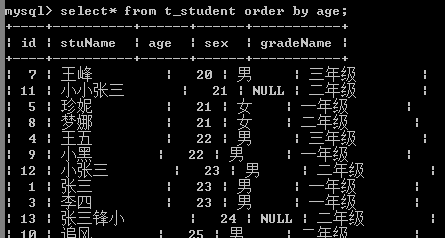
去重复查询：

* select distinct gradename from t\_student; （distinct关键词去重复）



对查询结果进行排序：最后可加ASC升序，或者DESC降序(默认升序)；

* select \* from t\_student order by age desc; （根据age大小进行排序）

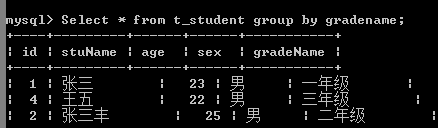


**distinct 用法细解：**实际中我们往往用distinct来返回不重复字段的条数（count(distinct id)）,其原因是distinct只能返回他的目标字段，而无法返回其他字段；使用类似select distinct name, id from user;语句时，distinct name, id 这样的命令，mysql 会认为要过滤掉name和id两个字段都重复的记录，如果mysql这样写：select id, distinct name from user；，mysql会报错，因为distinct必须放在要查询字段的开头。所以一般distinct用来查询不重复记录的条数。如果要查询不重复的记录，有时候可以用group by ：select id, name from user group by name;

Group by 分组查询：

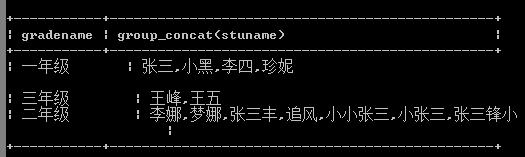
Group by 属性名 [HAVING条件表达式][WITH ROLLUP]

* select \* from t\_student group by gradename; (单独使用毫无意义)



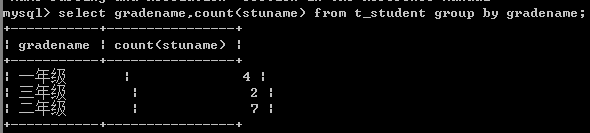
建议与group\_concat函数一起使用：(注意是逗号)

* select gradename, group\_concat(stuname) from t\_student group by gradename;



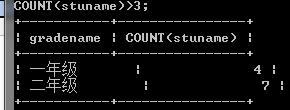
与聚合函数一起使用：

* select gradename, count(stuname) from t\_student group by gradename;



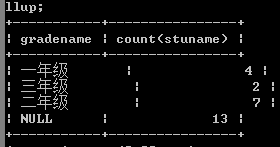
与HAVING一起使用，限制输出结果：

* select gradename, count(stuname) from t\_student group by gradename having count(stuname)>3;



与with rollup一起使用，在最后一行加一个总行和：

* select gradename, count(stuname) from t\_student group by gradename with rollup;



注：还可以与order by函数一起使用，进行排序；

* select 类别, 摘要, sum(数量) as 数量之和 from A group by all 类别, 摘要；



* select 类别, 摘要, sum(数量) AS 数量之和 from A group by 类别, 摘要；

(上式还可以这样写，推荐这样写)

**where 与having的区别：**

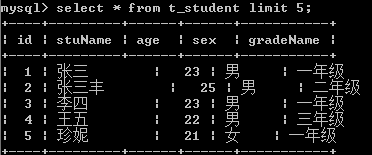
where 子句的作用是在对查询结果进行分组前，将不符合where条件的行去掉，即在分组之前过滤数据，where条件中不能包含聚组函数，使用where条件过滤出特定的行。having 子句的作用是筛选满足条件的组，即在分组之后过滤数据，条件中经常包含聚组函数，使用having 条件过滤出特定的组，也可以使用多个分组标准进行分组。

* select 类别, sum(数量) as 数量之和 from A group by 类别

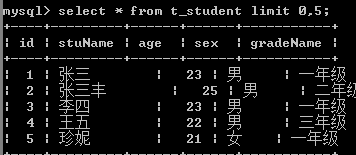
having sum(数量) > 18；（上面叙述的例句）

Limit分页查询：

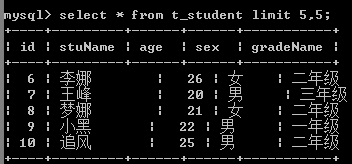
* select \* from t\_student limit 5; (输出5条信息)



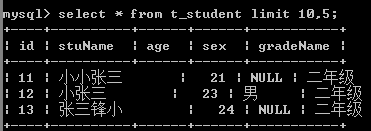
* select \* from t\_student limit 0,5; （和上面那条信息一样的作用）



* select \* from t\_student limit 5,5; （从6开始输出5条信息）



* select \* from t\_student limit 10,5; （从11开始输出最多5条信息）



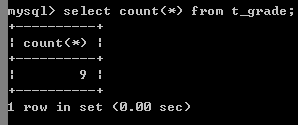
注：LIMIT 子句可以被用于强制 SELECT 语句返回指定的记录数。LIMIT 接受一个或两个数字参数。参数必须是一个整数常量。如果给定两个参数，第一个参数指定第一个返回记录行的偏移量，第二个参数指定返回记录行的最大数目。初始记录行的偏移量是 0(而不是 1)。

**使用聚合函数查询：**

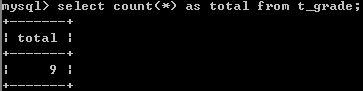
Count()函数：

* select count(\*) from t\_grade; （计算的是表中有多少条数据）

（t\_grade是表名，且count（\*）与count（N），N>0的效果相同）

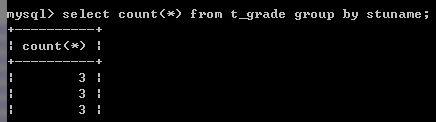


* select count(\*) as total from t\_grade;（通过as给count(\*)赋予新的变量名）

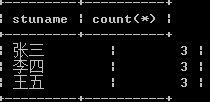


与group by 一起使用：

* select count(\*) from t\_grade group by stuname;



* select stuname, count(\*) from t\_grade group by stuname;



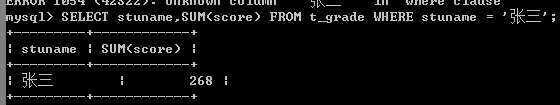
* select stu\_sex, stu\_class, count(\*) as count from tb\_student group by

stu\_sex, stu\_class;（这和上面的group by的用法差不多）

Sum函数，求和：

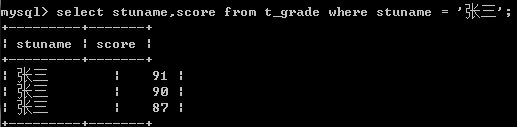
* select stuname, sum(score) from t\_grade where stuname = ‘张三’;

（这条命令我用下面这个命令细细解释，先看效果，如下：）



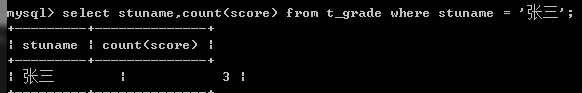
* select stuname, score from t\_grade where stuname = `张三`;

（查出张三的分数，结合上面的命令理解，应该就知道了）

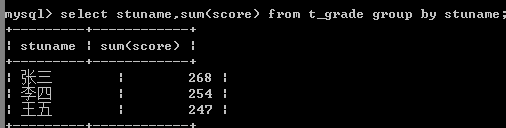


* select stuname, count(score) from t\_grade where stuname = `张三`;

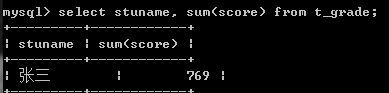
(sum 是加在一起，count是计算有多少条)



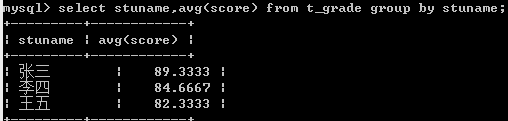
* select stuname, sum(score) from t\_grade group by stuname;



* select stuname, sum(score) from t\_grade; (用来和上面这条命令区别)

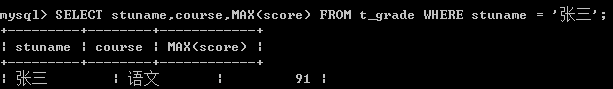


- select stuname, avg(score) from t\_grade group by stuname;

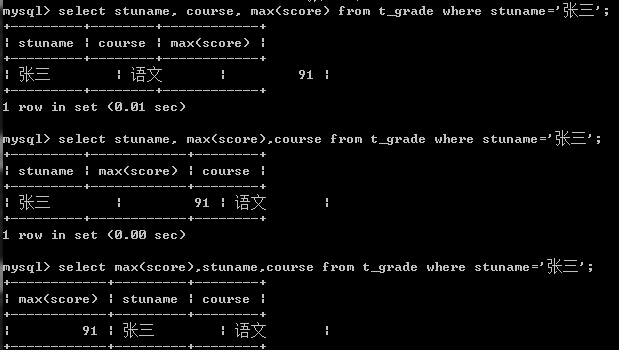


Max函数：

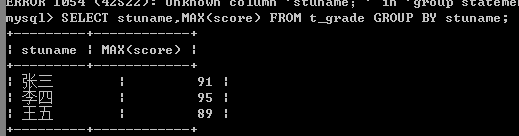
* select stuname, course, max(score) from t\_grade where stuname = ‘张三’；



（是查找出张三的所有课程中分数最高的那门，并返回选取的字段名称，然后看看顺序变换后是否会影响查询结果）



* select stuname, max(score) from t\_grade group by stuname；

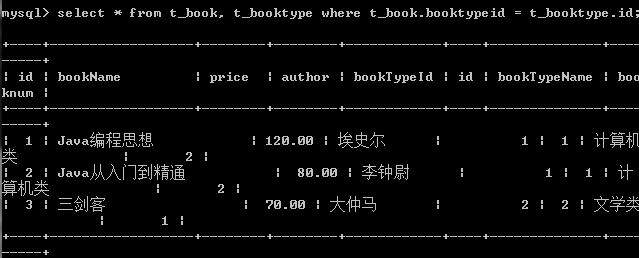


Min函数和max函数类似：

链接查询：(注意是英文状态下句号.)

* select \* from t\_book, t\_booktype where t\_book.booktypeid = t\_booktype.id;

（这里有两个表，t\_book以及t\_booktype；然后返回满足条件的数据）

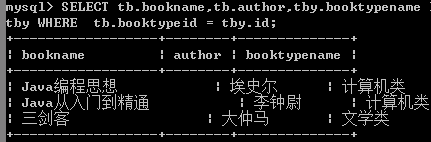


（这只是显示问题）

* select bookname, author, booktypename from t\_book, t\_booktype where t\_book.booktypeid = t\_booktype.id;



* select tb.bookname, tb.author, tby.booktypename from t\_book tb, t\_booktype tby where tb.booktypeid = tby.id; （这是给表赋变量名）



外连接查询：(也可以用别名的)

注：a）LEFT  JOIN或LEFT OUTER JOIN （效果一样的）：左向外联接的结果集包括  LEFT OUTER子句中指定的左表的所有行，而不仅仅是联接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有选择列表列均为空值；

b) RIGHT  JOIN 或 RIGHT  OUTER  JOIN  右向外联接是左向外联接的反向联接。将返回右表的所有行。如果右表的某行在左表中没有匹配行，则将为左表返回空值；

下图为t\_book以及t\_bootype两张表的内容：



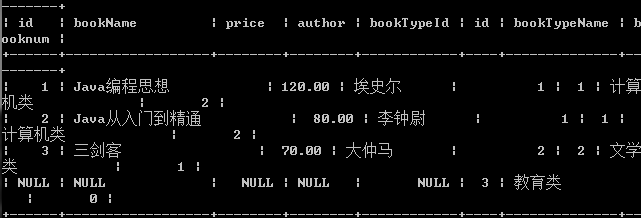
左连接：

* select \* from t\_book left join t\_booktype on t\_book.booktypeid = t\_booktype.id;



右链接：

* select \* from t\_book right join t\_booktype on t\_book.booktypeid = t\_booktype.id;



多条件查询：

* select tb.bookname, tb.author,tby.booktypename from t\_book tb, t\_booktype tby where tb.booktypeid=tby.id and tb.price > 70;



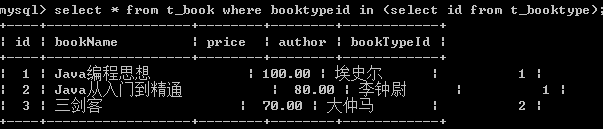
子查询：

带in关键字的子查询：

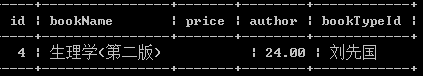
* select id from t\_booktype;



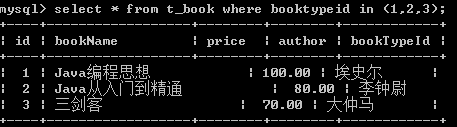
* select \* from t\_book where booktypeid in (select id from t\_booktype);



* select \* from t\_book where booktypeid not in (select id from t\_booktype);



* select \* from t\_book where booktypeid in (1,2,3);



* select price from t\_pricelevel where pricelevel=1;



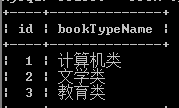
* select \* from t\_book where price >=(select price from t\_pricelevel where pricelevel=1);



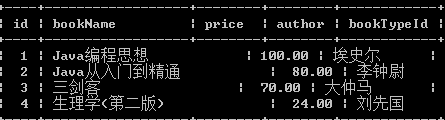
带exists关键字的子查询：（exists的返回结果是bool型，只有true或者false）

注：IN适合于外表数据量大而内表数据小的情况；EXISTS适合于外表小而内表大的情况；exists与not exists是相对应的。

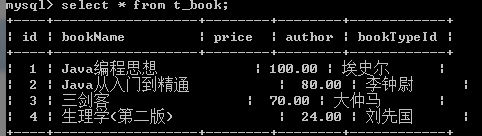
* select \* from t\_booktype;



* select \* from t\_book where exists (select \* from t\_booktype);



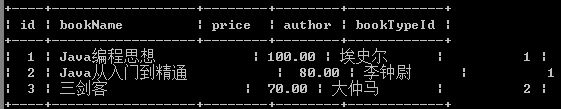
* select \* from t\_book;



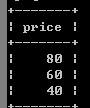
Any关键字：

* select \* from t\_book where price >= any(select price from t\_pricelevel);

(大于任何一个条件就好了)

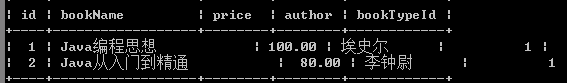


- select price from t\_pricelevel;



带all的关键字子查询：

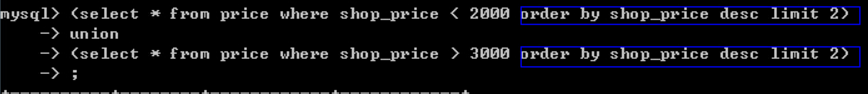
* select \* from t\_book where price >= all(select price from t\_pricelevel);



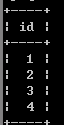
合并查询：

UNION：(合并不重复的部分) UNION 用于合并两个或多个 SELECT 语句的结果集，并消去表中任何重复行。UNION 内部的 SELECT 语句必须拥有相同数量的列，列也必须拥有相似的数据类型。同时，每条 SELECT 语句中的列的顺序必须相同。UNION 结果集中的列名总是等于第一个 SELECT 语句中的列名。

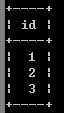
如果要用order by，需要加上括号，且使用limit关键字才可以达到排序效果；如下：



* select id from t\_book;



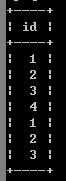
* select id from t\_booktype;



* select id from t\_book union select id from t\_booktype;

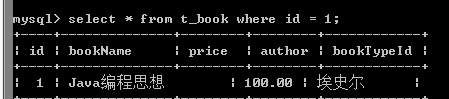


* select id from t\_book union all select id from t\_booktype;(union all不去掉重复的)

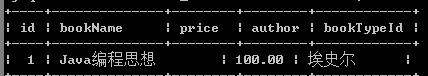


为表和字段取别名：

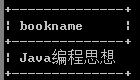
* select \* from t\_book where id = 1;



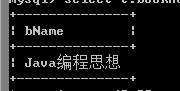
* select \* from t\_book t where t.id = 1; （加上别名之后效果一样）



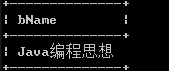
* select t.bookname from t\_book t where t.id = 1; （mysql似乎是先看别名的）



* select t.bookname bName from t\_book t where t.id = 1; （给字段赋别名）



* select t.bookname as bName from t\_book t where t.id = 1;



插入数据：

给表的所有字段加入数据：

* insert into t\_book values(null, ‘我爱我家’, 20, ‘张三’, 1);



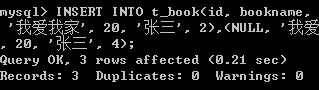
给表的指定字段插入数据：

* insert into t\_book (id, bookname, price, author, booktypeid) values(null, ‘我爱我家’, 20, ‘张三’, 1);



同时插入多条记录：

* insert into t\_book(id, bookname, price, author, booktypeid) values (null, '我爱我家', 20, '张三', 2),(null, '我爱我家', 20, '张三', 3), (null, '我爱我家', 20, '张三', 4); （多条记录的话用逗号隔开就好）

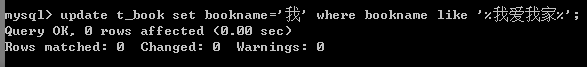


更新数据：

* update t\_book set bookname='java编程思想',price =120 where id=1;(价格修改为120)



* update t\_book set bookname=‘我’ where bookname like ‘%我爱我家%’; (修改为我)



删除数据：

* delete from t\_book where id=5;



* delete from t\_book where bookname = ‘我’;



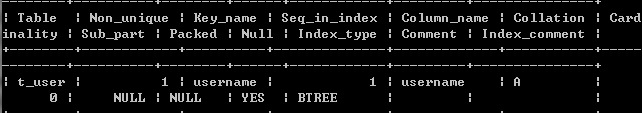
**索引的引入：**其作用是提高多表中数据的查询速度，类似于图书目录：

优点：提高数据查询速度，缺点：创建和维护索引的时间增加了；

**索引的分类：**

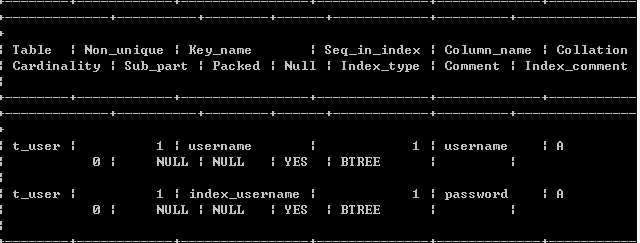
**1，**普通索引：这类索引可以创建在任何数据类型中；

* create table t\_user (id int, username varchar(20), password varchar(20), **index** (username));(在创建表的时候创建索引，默认索引名为username)
* show index from t\_user; (查看刚才创建的索引)



* create **index** index\_username on t\_user(username); （index\_username是索引名）

（在已经创建的表中设置索引）



* alter table t\_user add **index** index\_username(username);

（在已经创建的表中设置索引）

**2，**唯一性索引：**(主键默认是唯一性索引)**

创建唯一性索引用Unique参数设置；

* create table t\_user (id int, username varchar(20), password varchar(20), **unique index** (username));**(是在创建表的时候顺便创建索引)（结果如下图）**

或者：

* create table t\_user (id int, username varchar(20), password varchar(20), **unique index** index\_username(username)); （在这里不展现结果了）

(这样是加别名)(其他类型也可以加别名，不一一列举)



* create **unique index** index\_username on t\_user(username);
* alter table t\_user add **unique** **index** index\_username(username);

**3，**全文索引：

使用FULLTEXT参数可以设置，全文索引只能在创建CHAR，VARCHAR，TEXT类型的字段上，主要作用是提高查询较大字符串类型的速度，只有MyISAM引擎支持该索引，MYSQL默认引擎不支持；

**4，**单列索引：

在表中给单个字段设置索引，单列索引可以是普通索引，也可以是唯一索引，还可以是全文索引；

**5，**多列索引

多列索引在表中多个的字段上创建一个索引；

* create table t\_user (id int, username varchar(20), password varchar(20), **index** (username， password));
* create **index** index\_username on t\_user(username， password);
* alter table t\_user add **index** index\_username(username, password);

**6，**空间索引：

使用SPATIAL参数可以设置空间索引，空间索引只能建立在空间数据类型上，这样可以提高系统获取空间数据的效率；

**删除索引：**

* drop **index** index\_username on t\_user;

**视图：**

**视图定义**：视图是指计算机数据库中的视图，是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，视图并不在数据库中以存储的数据值集形式存在，即不是真实存在的。行和列数据来自由定义视图的查询所引用的表，并且在引用视图时动态生成。简单的来说视图是由其定义结果组成的表；当数据表结构很复杂，但我们只关心其中一部分数据的时候就可以使用视图，定义关心的数据；

**优点**：使操作简便化；增加数据的安全性；提高表的逻辑独立性；

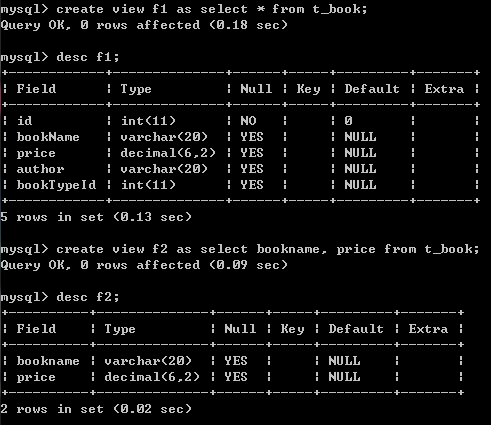
**视图的操作**：

|  |  |
| --- | --- |
| 操作指令 | 代码 |
| 创建视图 | CREATE VIEW 视图名(列1，列2...) AS SELECT (列1，列2...) FROM ...; |
| 使用视图 | 当成表使用就好 |
| 修改视图 | CREATE OR REPLACE VIEW 视图名 AS SELECT [...] FROM [...]; |
| 查看数据库已有视图 | >SHOW TABLES [like...];（可以使用模糊查找） |
| 查看视图详情 | DESC 视图名或者SHOW FIELDS FROM 视图名 |

创建视图：

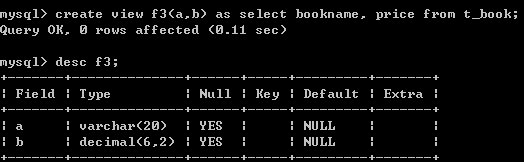
单表创建：

* create view v1 as select \* from t\_book;
* create view v2 as select bookname, price from t\_book;



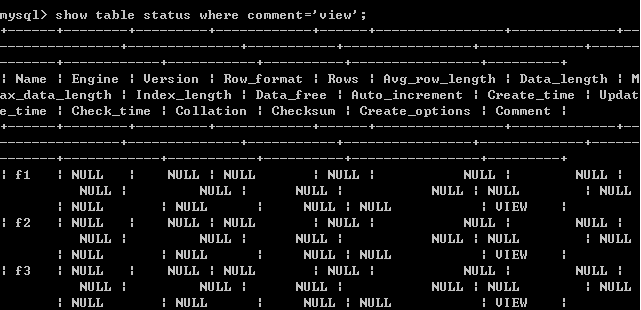
* create view v3(b,p) as select bookname, price from t\_book;

**（注：我的例句可能和截图不一样，但格式都是一样的，因为我有进行过其他操作，而且没有展示出来）**

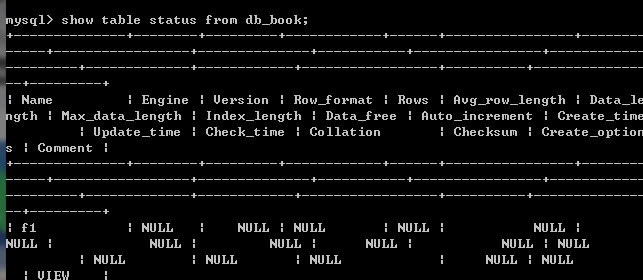


**注：查看数据库中的所有视图：**

* **show table status where comment=’view’;**

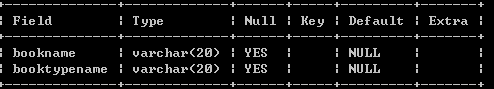


* **show table status from tarena;(能找到tarena数据库实例的所有表格和视图)**



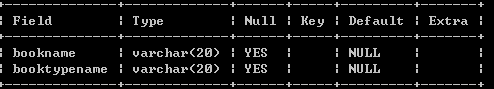
多表创建：

* create view v4 as select bookname, booktypename from t\_book, t\_booktype where t\_book.booktypeid = t\_booktype.id; (建议用别名)

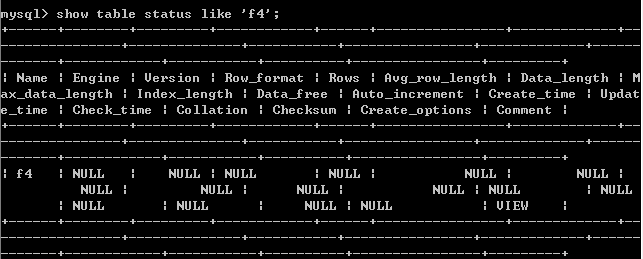


查看视图：

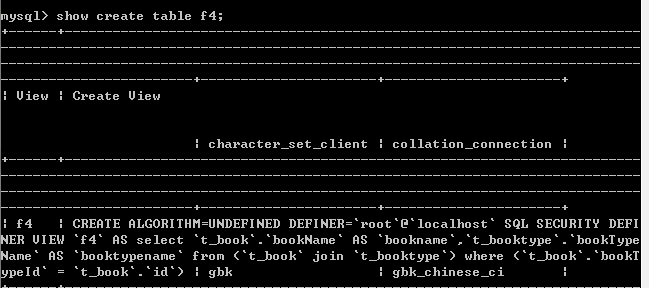
* desc f4；



* show table status like ‘f4’;(也可以查看表格，替换v4即可)

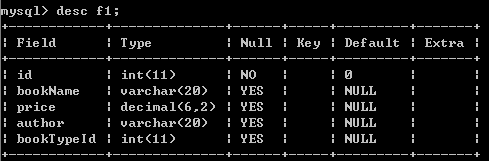


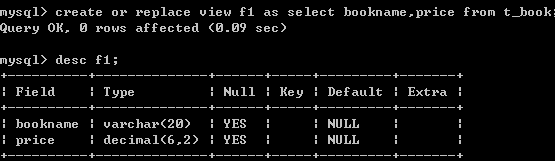
* show create view f4; (建视图的详细信息)



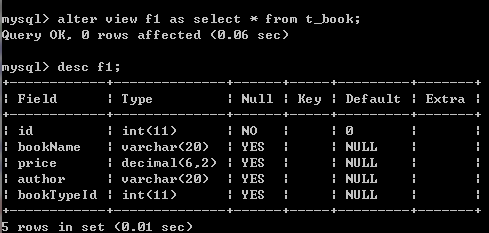
修改视图：

* create or replace view v1(bookname, price) as select bookname, price from t\_book; (or是必须的，不是让你选一个，先看看f1视图，然后看看修改之后的)





* alter view f1 as select \* from t\_book; （下图为再次修改之后）

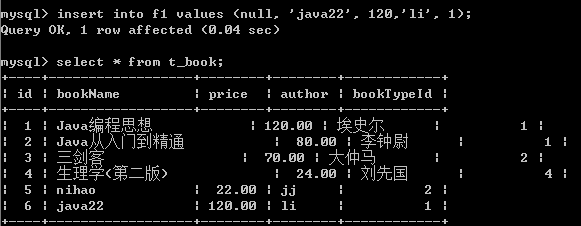


更新视图：

注：更新视图是通过视图来插入INSERT，更新UPDATE，删除DELETE表中的数据。因为，视图是一个虚拟表，其中没有数据，所以，当通过视图更新数据时，其实，是在更新基本表中的数据，如果对视图中的数据进行增加，或者删除操作时，实际上是在对其基本表中的数据，进行增加或者删除操作。

插入：

* insert into v1 values (null, ‘java good’, 120, ‘feng’, 1);



（上面截图中的第一个命令是设置自增长键按照要求的开始增长）

更新：

* update v1 set bookname = ‘very good’, price=200 where id =1;



删除视图：

注：删除视图时并不会删除数据，只是把视图的定义给删除了；

* drop view if exists viewname;



**触发器：**

注：触发器（trigger）是由事件来处罚某个操作，这些事件包括insert语句，update语句，delete语句，当数据库系统之星这些事件时，就会激活触发器执行相应的操作；

创建与使用触发器：(过渡变量new代表新增加的，old代表删除的)

单个执行语句：

* create trigger 触发器名字 before|after 触发事件 on 表名 for each row 执行的语句；（这是标准的单个操作的模板）
* create trigger trig\_book after insert on t\_book for each row update t\_booktype set bookname = booknum+1 where new.t\_booktypeid = t\_booktype.id;

多个执行语句：

* create trigger 触发器名字 before|after 触发事件 on 表名 for each row begin执行的语句列表 end；（这是标准的多个操作的模板）

**注：**一般情况下，mysql遇到分好；即会执行，但有时你想让多个语句一起执行，但这些语句中有分号，你就可以使用delimiter了。然后回车，那么MySQL将立即执行该语句。

delimiter是分隔符的意思。mysql的默认分隔符是分号，但delimiter //可以改变 MySQL delimiter 为 “//”，遇见下一个“//”后就会执行两个“//”之间的语句，最后还要有一个delimiter ；语句，把分割后设置回默认的分号。

* Delimiter |

create trigger trig\_book2 after delete

On t\_book for each row

Begin

update t\_booktype set bookname = booknum+1 where new.t\_booktypeid =t\_booktype.id;

insert into t\_log values(null, now(), ‘在表里删除了一条数据’);

delete from t\_log where old.booktypeid = t\_test.id;

end

| delimiter ；

**注：NEW 与 OLD 详解：**上述示例中使用了NEW关键字，和 MSSQL中的INSERTED 和 DELETED 类似，MySQL 中定义了 NEW 和 OLD，用来表示触发器的所在表中，触发了触发器的那一行数据。具体地：在 INSERT 型触发器中，NEW 用来表示将要（BEFORE）或已经（AFTER）插入的新数据；在 UPDATE 型触发器中，OLD 用来表示将要或已经**被修**改的原数据，NEW 用来表示**将要或已经修改**为的新数据；在 DELETE 型触发器中，OLD 用来表示**将要或已经被**删除的原数据；使用方法： NEW.columnName （columnName 为相应数据表某一列名）另外，OLD 是只读的，而 NEW 则可以在触发器中使用 SET 赋值，这样不会再次触发触发器，造成循环调用（如每插入一个学生前，都在其学号前加“2013”）。

* show triggers；（和查看数据库（show databases;）查看表格（show tables;））

删除触发器：

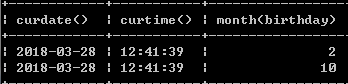
* drop trigger trig\_book；（或者是下面这个语句）
* drop trigger if exists 触发器名字；

**MYSQL函数：**

第一节：日期与时间函数：

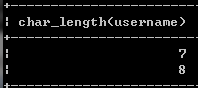
* curdate();返回当前日期；
* curtime（）；返回当前时间；
* month（）；返回日期中的月份；

- select curdate(), curtime(), month(birthday) from t\_t;



第二节：字符串函数：

* char\_length(s);计算字符串s的函数；
* upper（s）；把所有字母编程大写字母；
* lower（s）；你懂得
* select char\_length(username) from t\_t;

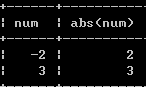


* select char\_length(username), upper (username), lower(username) from t\_t;

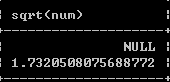


数学函数：

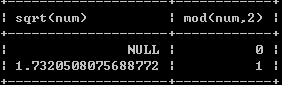
* abs(x);求绝对值；
* sqrt（x）；求平方根；
* mod（x）；求余数；
* select num, abs(num) from t\_t;



* select sqrt(num) from t\_t;



* select sqrt(num), mod(num,2) from t\_t;



加密函数：

* password（str）；一般对用户密码加密，不可逆；
* md5（str）；普通加密，不可逆；
* encode（str，pswd\_str）；加密函数，结果是一个二进制数，必须使用blob类型的字段进行保存；
* decode（ctrypt\_str，pswd——str）；解密函数
* insert into t\_t values(null, ‘2013-1-1’, ‘a’, 1, password(‘123456’));

(如果出现1406错误，可以把password的长度设置的再长一点)



* insert into t\_t values (null, ‘2014-10-3’, ‘a’, 1, md5(‘12344’), encode(‘abcd’,’aa’));



* select decode(pp, 'aa') from t\_t where id=5; (pp为所在列)



**存储过程与函数：**

第一节：存储过程与函数的引入：

注：存储过程与存储函数是数据库中定义的一些sql语句的集合，然后直接调用这些存储过程和函数来执行已经定义好的 sql语句，存储过程与函数可以避免开发人员重复的编写相同的sql语句，而且，存储过程与函数是在mysql服务器中存储和执行的，可以减少客户端和服务端的数据传输。

**创建存储过程：**

创建一个存储过程

* delimiter &&

create procedure pro\_book (in bt int, out count\_num int )

begin

select count(\*) from t\_book where booktypeid = bt;

end

&&

delimiter ; (注意delimiter与；之间有一个空格)

调用存储过程：

* Call pro\_book(1,@total);



查看存储过程：

* show procedure status；（可以加like查看具体的细节，或者用show create procedure + 过程名来查看细节）



创建一个存储函数：

创建存储函数：

* DELIMITER &&

create function function\_book (bookid int)

returns varchar(20)

begin

return (select bookname from t\_book where id=bookid);

end

&&

DELIMITER ;

* select function\_book(1);



**变量的使用：**

定义变量：declare var\_name type[default value];

为变量赋值：set var\_name= new\_value;

* DELIMITER &&

create procedure pre\_user()

begin

declare a, b varchar(20);

insert into t\_user values (null, a, b);

end

&&

DELIMITER ;

* call pre\_user();



设置变量；set

* DELIMITER &&

create procedure pre\_user2()

begin

declare a, b varchar(20);

set a = 'java1234',b='123456';

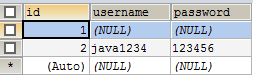
insert into t\_user values (null, a, b);

end

&&

DELIMITER ;

* call pre\_user2();



或者：

* DELIMITER &&

create procedure pre\_user3()

begin

declare a, b varchar(20);

select username2,password2 into a, b from t\_user2 where id2=1; (也是赋值的一个方法)

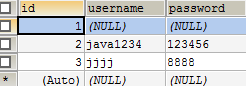
insert into t\_user values (null, a, b);

end

&&

DELIMITER ;

* call pre\_user3();



**游标的使用：**

注：查询语句可能查询出多条记录，在存储过程中和函数中使用游标来逐条读取查询结果集中的记录，游标的使用包括声明游标，打开游标，使用游标和关闭游标，游标必须声明在处理程序之前，并且声明在变量和条件之后。

**声明游标：**

* DELIMITER &&

create procedure pre\_user5()

begin

declare a, b varchar(20);

declare cur\_t\_user2 cursor for select username2, password2 from t\_user2; #声明游标

open cur\_t\_user2; #打开游标

fetch cur\_t\_user2 into a, b; #使用游标

insert into t\_user values (null, a, b);

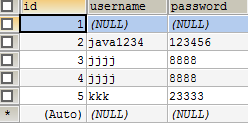
close cur\_t\_user2; #关闭游标

end

&&

DELIMITER ;

* call pre\_user5();



**注变量**：局部变量一般用在sql语句块中，比如存储过程的begin/end。其作用域仅限于该语句块，在该语句块执行完毕后，局部变量就消失了；用户变量的作用域要比局部变量要广。用户变量可以作用于当前整个连接，但是当当前连接断开后，其所定义的用户变量都会消失。用户变量使用如下(这里我们无须使用declare关键字进行定义，可以直接这样使用):select @变量名

局部变量一般用declare来声明，可以使用default来说明默认值

**流程控制：**

流程控制的使用：在存储过程和函数中可以使用流程控制来控制语句的执行，Mysql中使用if语句，case语句，loop语句，leave语句，iterate语句，repeat语句和while语句进行流程控制。

if语句：

* DELIMITER &&

create procedure pro\_user6(in bookid int)

begin

select count(\*) into @num from t\_user where id = bookid;

if @num > 0 then update t\_user set username='java12345' where id = bookid;

else insert into t\_user values(null, '23332', '32223');

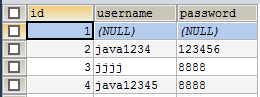
end if;

end

&&

DELIMITER ;

* call pro\_user6(7);



case 语句：

* DELIMITER &&

create procedure pro\_user7(in bookid int)

begin

select count(\*) into @num from t\_user where id = bookid;

case @num

when 1 then update t\_user set username='java1111' where id = bookid;

when 2 then insert into t\_user values(null, '2222111', '999');

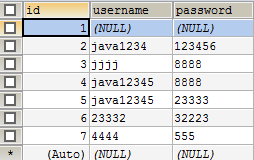
else insert into t\_user values(null, '4444', '555');

end case ;

end

&&

DELIMITER ; （不call，直接展示）



Loop语句：

注：loop语句可以使某些特定的语句重复执行，实现一个简单的循环。但是loop语句本身没有停止循环的语句，必须是遇到leave语句等才可以停止循环，loop语句的语法的基本形式如下：

loop语句

* DELIMITER &&

create procedure pro\_user8(in totalnum int)

begin

aaa:loop （前面是loop的标签）

set totalnum = totalnum - 1;

if totalnum = 0 then **leave aaa**;

else insert into t\_user values(totalnum, '888', '6666');

end if ;

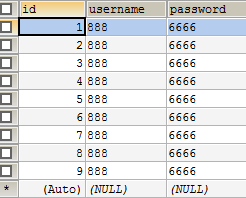
end loop aaa;

end

&&

DELIMITER ;

* call pro\_user8(10);



Iterate 用法：

1，iterate语句：iterate语句也是跳出循环的语句，但是，iterate语句是跳出本次循环，然后直接进入下一次循环，基本语法：

* DELIMITER &&

create procedure pro\_user10(in totalnum int)

begin

aaa:loop

set totalnum = totalnum - 1;

if totalnum = 0 then leave aaa;

**elseif totalnum = 3 then iterate aaa;**

end if ;

insert into t\_user values(totalnum, '558', '00');

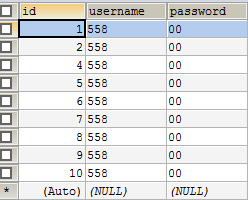
end loop aaa;

end

&&

DELIMITER ;

* call pro\_user10(11);



Repeat语句：repeat语句是有条件控制的循环语句，当满足条件时，就会跳出循环语句，repeat语句的基本语法形式如下：

* DELIMITER &&

create procedure pro\_user11(in totalnum int)

begin

repeat

set totalnum = totalnum-1;

insert into t\_user values(totalnum, '098', '765');

until totalnum = 1

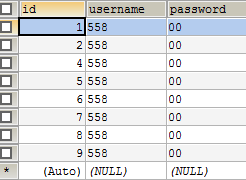
end repeat;

end

&&

DELIMITER ;

* call pro\_user10(10);



While语句：

* DELIMITER &&

create procedure pro\_user12(in totalnum int)

begin

while totalnum > 0 do

insert into t\_user values(totalnum, '456', '654');

set totalnum = totalnum-1;

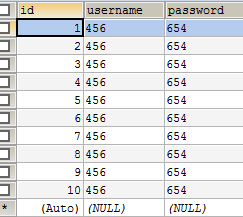
end while;

end

&&

DELIMITER ;

* call pro\_user12(10);



**调用存储过程函数：**

**查看存储过程和函数：**

**1，**第一种方法：

* show procedure status like 'pro\_book';



**2**，第二种方法：

* show create procedure pro\_book;

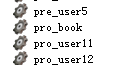


**3，**第三种方法：



**删除存储过程函数:**

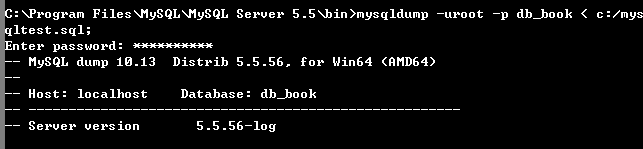
* drop procedure pro\_user10;



**数据的备份与还原：是在cmd里执行下面的命令的，在mysql/bin文件夹里打开cmd；**

导出整个数据库(包括数据库中的数据）

* mysqldump -u username -p dbname > dbname.sql；（可以是一个路径）



导出数据库结构（不含数据）

* mysqldump -u username -p -d dbname > dbname.sql；

导出数据库中的某张数据表（包含数据）

* mysqldump -u username -p dbname tablename > tablename.sql；

导出数据库中的某张数据表的表结构（不含数据）

* mysqldump -u username -p -d dbname tablename > tablename.sql；
* all-databases , -A 导出全部数据库. mysqldump -uroot -p --all-databases > all\_databases\_backup.sql;
* all-tablespaces , -Y 导出全部表空间。mysqldump -uroot -p –all-databases > all\_tablespaces.sql;

恢复数据库的命令：

* mysql -u username -p test\_db < test\_db.sql ；