MYSQL基础语法

SQL 是用于访问和处理数据库的标准的计算机语言。

常用命令

```
SELECT #从数据库中提取数据
UPDATE #更新数据库中的数据
DELETE #从数据库中删除数据
INSERT INTO #向数据库中插入新数据
CREATE DATABASE #创建新数据库
ALTER DATABASE #修改数据库
CREATE TABLE #创建新表
ALTER TABLE #变更（改变）数据库表
DROP TABLE #删除表
CREATE INDEX #创建索引（搜索键）
DROP INDEX #删除索引
```

数据类型

数值类型：

类型	大小	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	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 字节	(-9,223,372,036,854,775,808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值
FLOAT	4 字节	(-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8 字节	(-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度浮点数值
DECIMAL	对DECIMAL(M,D)，如果M>D，为M+2否则为D+2	依赖于M和D的值	依赖于M和D的值	小数值

日期类型：

类型	大小(字节)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2038 结束时间是第 **2147483647** 秒，北京时间 **2038-1-19 11:14:07**，格林尼治时间 2038年1月19日 凌晨 03:14:07	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

文本类型：

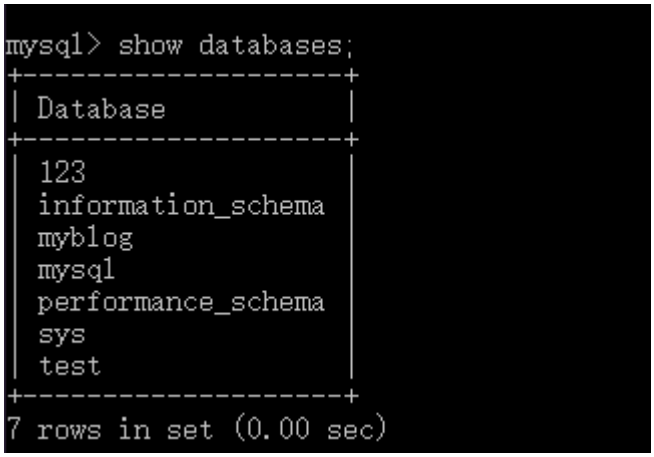
类型	大小	用途
CHAR	0-255字节	定长字符串
VARCHAR	0-65535 字节	变长字符串
TINYBLOB	0-255字节	不超过 255 个字符的二进制字符串
TINYTEXT	0-255字节	短文本字符串
BLOB	0-65 535字节	二进制形式的长文本数据
TEXT	0-65 535字节	长文本数据
MEDIUMBLOB	0-16 777 215字节	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215字节	中等长度文本数据
LONGBLOB	0-4 294 967 295字节	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295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

数据库表

一个数据库通常包含一个或多个表。

查看有哪些库：

```
show databases;
```

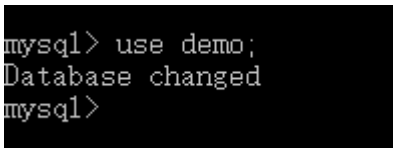


创建数据库:

```
create databases name;
```

创建表之前，需要先使用数据库。

```
use demo; #表示使用demo数据库
```



查看当前库下有什么表。

```
show tables;
```

删除数据库

```
drop database 数据库名;
```

表操作

```
CREATE TABLE table_name
(
column_name1 data_type(size),
column_name2 data_type(size),
column_name3 data_type(size),
....
)
#column_name 参数规定表中列的名称。
#data_type 参数规定列的数据类型（例如varchar/integer/decimal、date 等等）。
#size 参数规定表中列的最大长度。
```

约束条件

```
在创建是，也可以在数据类型的后面加上约束条件
NOT NULL #使用非空约束，表示一个字段的内容不允许为空，即插入数据时必须插入
UNIQUE #表示一个字段的内容不允许重复
PRIMARY KEY #主键(不为空、唯一)
FOREIGN KEY #外键
```

PRIMARY KEY 约束唯一标识数据库表中的每条记录。每个表只能定义一个主键。

主键值必须唯一标识表中的每一行，且不能为 NULL，即表中不可能存在两行数据有相同的主键值。这是唯一性原则。在 CREATE TABLE 语句中，主键是通过 PRIMARY KEY 关键字来指定的。

在定义列的同时指定主键，语法规则如下：

比如创建一个学生表：

```
create table student
(
sno varchar(20) primary key,
sname varchar(20) not null,
ssex varchar(20) not null,
sbirthday datetime,
class varchar(20)
);
```

查看表结构

```
DESC table_name #可以查看表结构。
```

删除数据表：

```
DROP TABLE 数据表名;
```

数据操作：

表创建好了只后就可以往里面插入数据了。使用INSERT INTO向表中插入数据。基本格式如下：

```
#基本语法
INSERT INTO table_name VALUES (value1,value2,value3,...);
#示例
insert into student values('108','曾华','男','1977-09-01','95033');
```

删除数据：

```
DELETE FROM table_name WHERE Clause #删除表table_name里面满足clause的条件
#示例
DELETE FROM student WHERE sno='108' #删除表table_name里面满足clause的条件
```

修改数据：

```
UPDATE table_name SET field1=new-value1, field2=new-value2 [WHERE Clause];
#示例
UPDATE STUDENT SET ssex='女' where sno =108 #将sno=108的ssex修改为女
```

查询数据:SELECT语句

```
#基本语法
SELECT column_name,column_name FROM table_name; #获取指定列数据
#常用用法
SELECT * FROM table_name #获取所有列数据
```

SQL进阶

在创建的学生表中在插入以下几条数据：

```
insert into student values('105','匡明','男','1975-10-02','95031');
insert into student values('107','王丽','女','1976-01-23','95033');
insert into student values('101','李军','男','1976-02-20','95033');
insert into student values('109','王芳','女','1975-02-10','95031');
insert into student values('103','陆君','男','1974-06-03','95031');
```

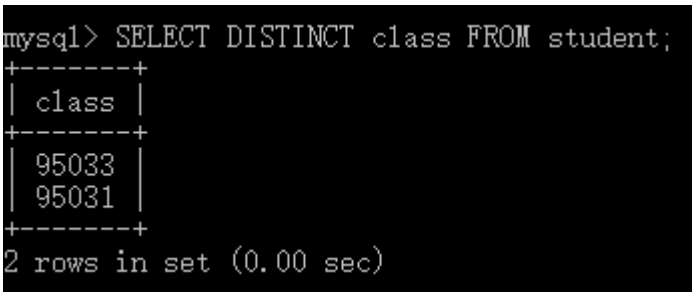
DISTINCT

在表中，一个列可能会包含多个重复值，有时您也许希望仅仅列出不同（DISTINCT）的值。

DISTINCT 关键词用于返回唯一不同的值。

查询有哪些班级：

```
SELECT DISTINCT class FROM student;
```



WHERE

条件查询，使用WHERE子句查询满足条件的数据。

```
SELECT column_name,column_name
FROM table_name
WHERE column_name operator value;
```

例如：查询性别为男的学生

```
SELECT * FROM student WHERE ssex='男';
```

```
mysql> SELECT * FROM student WHERE ssex='男';
+-----+-------+------+---------------------+-------+
| sno | sname | ssex | sbirthday           | class |
+-----+-------+------+---------------------+-------+
| 101 | 李军  | 男   | 1976-02-20 00:00:00 | 95033 |
| 103 | 陆君  | 男   | 1974-06-03 00:00:00 | 95031 |
| 105 | 匡明  | 男   | 1975-10-02 00:00:00 | 95031 |
| 108 | 曾华  | 男   | 1977-09-01 00:00:00 | 95033 |
+-----+-------+------+---------------------+-------+
4 rows in set (0.00 sec)
```

AND & OR

如果第一个条件和第二个条件都成立，则 AND 运算符显示一条记录。

如果第一个条件和第二个条件中只要有一个成立，则 OR 运算符显示一条记录。

```
#例：查询性别为女，并且是95031班的学生
SELECT * FROM student WHERE ssex='女' AND class=95031;
```

```
mysql> SELECT * FROM student WHERE ssex='女' AND class=95031;
+-----+-------+------+---------------------+-------+
| sno | sname | ssex | sbirthday           | class |
+-----+-------+------+---------------------+-------+
| 109 | 王芳  | 女   | 1975-02-10 00:00:00 | 95031 |
+-----+-------+------+---------------------+-------+
1 row in set (0.00 sec)
```

```
#例：查询性别为女，或者是95031班的学生
SELECT * FROM student WHERE ssex='女' OR class=95031;
```

```
mysql> SELECT * FROM student WHERE ssex='女' OR class=95031;
+-----+-------+------+---------------------+-------+
| sno | sname | ssex | sbirthday           | class |
+-----+-------+------+---------------------+-------+
| 103 | 陆君  | 男   | 1974-06-03 00:00:00 | 95031 |
| 105 | 匡明  | 男   | 1975-10-02 00:00:00 | 95031 |
| 107 | 王丽  | 女   | 1976-01-23 00:00:00 | 95033 |
| 109 | 王芳  | 女   | 1975-02-10 00:00:00 | 95031 |
+-----+-------+------+---------------------+-------+
4 rows in set (0.00 sec)
```

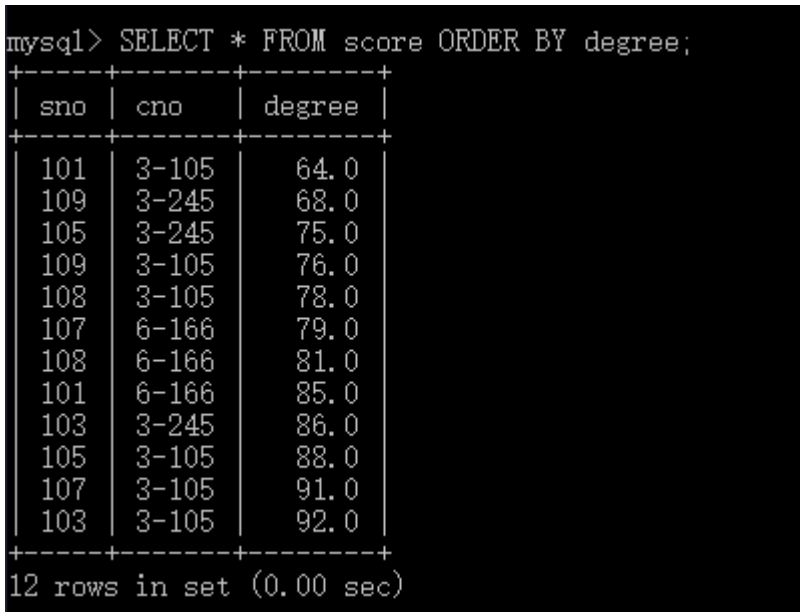
除了学生表，再创建以下几个表：并插入数据，可以直接复制粘贴。

```
#建立教师表
create table teacher
(
tno varchar(20) not null primary key,
tname varchar(20) not null,
tsex varchar(20) not null,
```

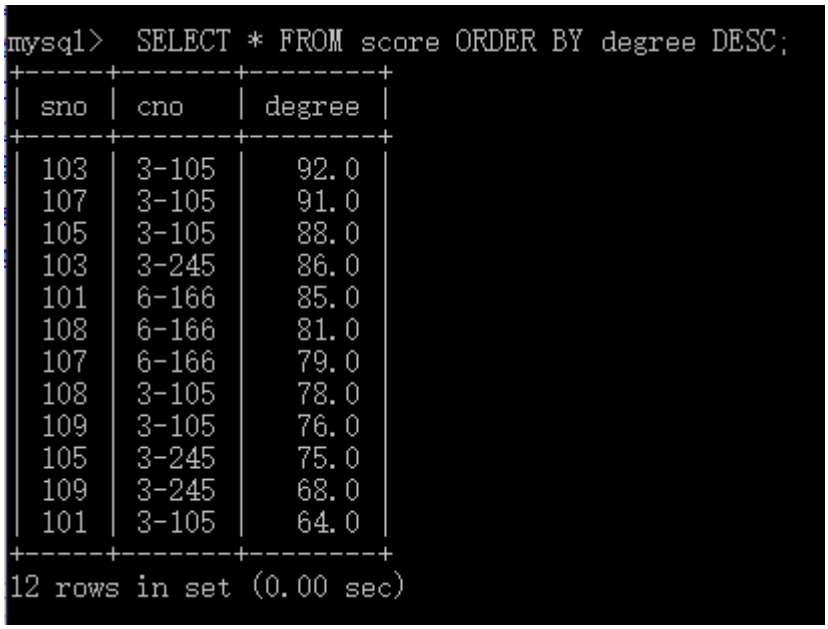
```
tbirthday datetime,
prof varchar(20),
depart varchar(20) not null

);
#建立课程表course
create table course
(
cno varchar(20) not null primary key,
cname varchar(20) not null,
tno varchar(20) not null,
foreign key(tno) references teacher(tno)

);
#建立成绩表
CREATE TABLE score(
sno varchar(20) not null,
foreign key(sno) references student(sno),
cno varchar(20) not null,
foreign key(cno) references course(cno),
degree decimal(4, 1),
primary key(sno, cno)
);

#添加教师表
insert into teacher values('804','李诚','男','1958-12-02','副教授','计算机系');
insert into teacher values('856','张旭','男','1969-03-12','讲师','电子工程系');
insert into teacher values('825','王萍','女','1972-05-05','助教','计算机系');
insert into teacher values('831','刘冰','女','1977-08-14','助教','电子工程系');

#添加课程表
insert into course values('3-105','计算机导论','825');
insert into course values('3-245','操作系统','804');
insert into course values('6-166','数字电路','856');
insert into course values('9-888','高等数学','831');

#添加成绩表
insert into score values('103','3-245','86');
insert into score values('105','3-245','75');
insert into score values('109','3-245','68');
insert into score values('103','3-105','92');
insert into score values('105','3-105','88');
insert into score values('109','3-105','76');
insert into score values('101','3-105','64');
insert into score values('107','3-105','91');
insert into score values('108','3-105','78');
insert into score values('101','6-166','85');
insert into score values('107','6-166','79');
insert into score values('108','6-166','81');
```

ORDER BY:排序

```
#例：在score表中查询数据，按升序排列
SELECT * FROM score ORDER BY degree;
```




```
#例：降序排序,会使用到DESC
SELECT * FROM score ORDER BY degree DESC;
```



GROUP BY

GROUP BY 语句根据一个或多个列对结果集进行分组。在分组的列上我们可以使用 COUNT, SUM, AVG,等函数。

```
例：在score表中求每门课程的平均分数
SELECT cno,AVG(degree) FROM score GROUP BY cno;
```

```
mysql> SELECT cno,AVG(degree) FROM score GROUP BY cno;
+-------+-------------+
| cno   | AVG(degree) |
+-------+-------------+
| 3-105 |    81.50000 |
| 3-245 |    76.33333 |
| 6-166 |    81.66667 |
+-------+-------------+
3 rows in set (0.00 sec)
```

```
#例：在student表中求每个班级人数
SELECT COUNT(*),class FROM student GROUP BY class;
```

```
mysql> SELECT COUNT(*),class FROM student GROUP BY class;
+----------+-------+
| COUNT(*) | class |
+----------+-------+
|        3 | 95033 |
|        3 | 95031 |
+----------+-------+
2 rows in set (0.00 sec)
```

HAVING

HAVING子句可以让我们筛选成组后的各种数据,where子句在聚合前先筛选记录，也就是说作用在group by和having子句前。而 having子句在聚合后对组记录进行筛选。

```
#例：在score表中，查询平均分大于80的课程
SELECT cno,AVG(degree) FROM score GROUP BY cno HAVING AVG(degree) > 80;
```

```
mysql> SELECT cno,AVG(degree) FROM score GROUP BY cno HAVING AVG(degree) > 80;
+-------+-------------+
| cno   | AVG(degree) |
+-------+-------------+
| 3-105 |    81.50000 |
| 6-166 |    81.66667 |
+-------+-------------+
2 rows in set (0.00 sec)
```

LIKE

模糊查询

```
#例：查询student表中姓王的同学
SELECT * FROM student WHERE sname LIKE '王%';
```

```
mysql> SELECT * FROM student WHERE sname LIKE '王%';
+-----+-------+------+---------------------+-------+
| sno | sname | ssex | sbirthday           | class |
+-----+-------+------+---------------------+-------+
| 107 | 王丽  | 女   | 1976-01-23 00:00:00 | 95033 |
| 109 | 王芳  | 女   | 1975-02-10 00:00:00 | 95031 |
+-----+-------+------+---------------------+-------+
2 rows in set (0.00 sec)
```

通配符:

%： 替代 0 个或多个字符

_ ： 替代一个字符

IN

IN 操作符允许您在 WHERE 子句中规定多个值。

```
#例：查询teacher表中职位为讲师和助教的数据
SELECT * FROM teacher WHERE prof IN ('讲师','助教');
```

```
mysql> SELECT * FROM teacher WHERE prof IN ('讲师','助教');
+-----+-------+------+---------------------+------+------------+
| tno | tname | tsex | tbirthday           | prof | depart     |
+-----+-------+------+---------------------+------+------------+
| 825 | 王萍  | 女   | 1972-05-05 00:00:00 | 助教 | 计算机系   |
| 831 | 刘冰  | 女   | 1977-08-14 00:00:00 | 助教 | 电子工程系 |
| 856 | 张旭  | 男   | 1969-03-12 00:00:00 | 讲师 | 电子工程系 |
+-----+-------+------+---------------------+------+------------+
3 rows in set (0.00 sec)
```

BETWEEN

BETWEEN 操作符选取介于两个值之间的数据范围内的值。这些值可以是数值、文本或者日期。

```
#例：在score表中查询分数介于70到80之间的学生
SELECT * FROM score WHERE degree BETWEEN 70 AND 80;
```

```
mysql> SELECT * FROM score WHERE degree BETWEEN 70 AND 80;
+-----+-------+--------+
| sno | cno   | degree |
+-----+-------+--------+
| 105 | 3-245 |   75.0 |
| 107 | 6-166 |   79.0 |
| 108 | 3-105 |   78.0 |
| 109 | 3-105 |   76.0 |
+-----+-------+--------+
4 rows in set (0.00 sec)
```

AS

通过使用 SQL，可以为表名称或列名称指定别名。

```
#例：查询“95033”班学生的平均分
SELECT AVG(degree) AS '平均分' FROM student AS st,score AS sc WHERE st.sno=sc.sno AND
st.class= 95033;
```

```
mysql> SELECT AVG(degree) AS '平均分' FROM student AS st,score AS sc WHERE st.sno=sc.sno AND st.class= 95033;
+----------+
| 平均分   |
+----------+
| 79.66667 |
+----------+
1 row in set (0.00 sec)
```

子查询

在查询中使用多个SELECT

```
#例：查询成绩比该课程平均成绩低的同学的成绩表。
SELECT * FROM score AS s1 WHERE s1.degree <
(SELECT AVG(s2.degree) FROM score AS s2 GROUP BY cno HAVING s1.cno =s2.cno );
```

```
mysql> SELECT * FROM score AS s1 WHERE s1.degree <
    -> (SELECT AVG(s2.degree) FROM score AS s2 GROUP BY cno HAVING s1.cno =s2.cno );
+-----+-------+--------+
| sno | cno   | degree |
+-----+-------+--------+
| 101 | 3-105 |   64.0 |
| 105 | 3-245 |   75.0 |
| 107 | 6-166 |   79.0 |
| 108 | 3-105 |   78.0 |
| 108 | 6-166 |   81.0 |
| 109 | 3-105 |   76.0 |
| 109 | 3-245 |   68.0 |
+-----+-------+--------+
7 rows in set (0.01 sec)
```

连接查询

join 用于把来自两个或多个表的行结合起来。常见联合方式LEFT JOIN、RIGHT JOIN、INNER JOIN

INNER JOIN

INNER JOIN 关键字在表中存在至少一个匹配时返回行。

```
#例：45、查询所有选修“计算机导论”课程的同学的成绩表。。
SELECT * FROM score
INNER JOIN course ON score.cno = course.cno AND course.cname = '计算机导论';
```

```
mysql> SELECT * FROM score
    ->
    -> INNER JOIN course
    ->
    -> ON score.cno = course.cno AND course.cname = '计算机导论';
+-----+-------+--------+-------+------------+-----+
| sno | cno   | degree | cno   | cname      | tno |
+-----+-------+--------+-------+------------+-----+
| 101 | 3-105 |   64.0 | 3-105 | 计算机导论 | 825 |
| 103 | 3-105 |   92.0 | 3-105 | 计算机导论 | 825 |
| 105 | 3-105 |   88.0 | 3-105 | 计算机导论 | 825 |
| 107 | 3-105 |   91.0 | 3-105 | 计算机导论 | 825 |
| 108 | 3-105 |   78.0 | 3-105 | 计算机导论 | 825 |
| 109 | 3-105 |   76.0 | 3-105 | 计算机导论 | 825 |
+-----+-------+--------+-------+------------+-----+
6 rows in set (0.00 sec)
```

LEFT JOIN

LEFT JOIN 从左表（table1）返回所有的行，即使右表（table2）中没有匹配。如果右表中没有匹配，则结果为 NULL。

```sql
SELECT * FROM student as s
LEFT JOIN
score as sc
ON s.sno = sc.sno AND sc.degree = 86;
```

```
mysql> SELECT * FROM student as s
    -> LEFT JOIN
    -> score as sc
    -> ON s.sno = sc.sno AND sc.degree = 86;
+-----+-------+------+---------------------+-------+------+------+--------+
| sno | sname | ssex | sbirthday           | class | sno  | cno  | degree |
+-----+-------+------+---------------------+-------+------+------+--------+
| 101 | 李军  | 男   | 1976-02-20 00:00:00 | 95033 | NULL | NULL |   NULL |
| 103 | 陆君  | 男   | 1974-06-03 00:00:00 | 95031 | 103  | 3-245|   86.0 |
| 105 | 匡明  | 男   | 1975-10-02 00:00:00 | 95031 | NULL | NULL |   NULL |
| 107 | 王丽  | 女   | 1976-01-23 00:00:00 | 95033 | NULL | NULL |   NULL |
| 108 | 曾华  | 女   | 1977-09-01 00:00:00 | 95033 | NULL | NULL |   NULL |
| 109 | 王芳  | 女   | 1975-02-10 00:00:00 | 95031 | NULL | NULL |   NULL |
+-----+-------+------+---------------------+-------+------+------+--------+
6 rows in set (0.00 sec)
```

RIGHT JOIN

RIGHT JOIN 从右表（table2）返回所有的行，即使左表（table1）中没有匹配。如果左表中没有匹配，则结果为 NULL。

```sql
SELECT * FROM student as s
RIGHT JOIN
score as sc
ON s.sno = sc.sno AND sc.degree = 86;
```

```
mysql> SELECT * FROM student as s
    ->
    -> RIGHT JOIN
    ->
    -> score as sc
    ->
    -> ON s.sno = sc.sno AND sc.degree = 86;
+------+-------+------+---------------------+-------+------+-------+--------+
| sno  | sname | ssex | sbirthday           | class | sno  | cno   | degree |
+------+-------+------+---------------------+-------+------+-------+--------+
| NULL | NULL  | NULL | NULL                | NULL  | 101  | 3-105 |   64.0 |
| NULL | NULL  | NULL | NULL                | NULL  | 101  | 6-166 |   85.0 |
| NULL | NULL  | NULL | NULL                | NULL  | 103  | 3-105 |   92.0 |
| 103  | 陆君  | 男   | 1974-06-03 00:00:00 | 95031 | 103  | 3-245 |   86.0 |
| NULL | NULL  | NULL | NULL                | NULL  | 105  | 3-105 |   88.0 |
| NULL | NULL  | NULL | NULL                | NULL  | 105  | 3-245 |   75.0 |
| NULL | NULL  | NULL | NULL                | NULL  | 107  | 3-105 |   91.0 |
| NULL | NULL  | NULL | NULL                | NULL  | 107  | 6-166 |   79.0 |
| NULL | NULL  | NULL | NULL                | NULL  | 108  | 3-105 |   78.0 |
| NULL | NULL  | NULL | NULL                | NULL  | 108  | 6-166 |   81.0 |
| NULL | NULL  | NULL | NULL                | NULL  | 109  | 3-105 |   76.0 |
| NULL | NULL  | NULL | NULL                | NULL  | 109  | 3-245 |   68.0 |
+------+-------+------+---------------------+-------+------+-------+--------+
12 rows in set (0.00 sec)
```

UNION

UNION 合并两个或多个 SELECT 语句的结果。

```
#例：查询所有“女”教师和“女”同学的name、sex和birthday。
SELECT sname,ssex,sbirthday FROM student
WHERE ssex= '女'
UNION
SELECT tname,tsex,tbirthday FROM teacher
WHERE tsex= '女';
```

```
mysql> SELECT sname,ssex,sbirthday FROM student
    -> WHERE ssex= '女'
    -> UNION
    -> SELECT tname,tsex,tbirthday FROM teacher
    -> WHERE tsex= '女';
+-------+------+---------------------+
| sname | ssex | sbirthday           |
+-------+------+---------------------+
| 王丽  | 女   | 1976-01-23 00:00:00 |
| 曾华  | 女   | 1977-09-01 00:00:00 |
| 王芳  | 女   | 1975-02-10 00:00:00 |
| 王萍  | 女   | 1972-05-05 00:00:00 |
| 刘冰  | 女   | 1977-08-14 00:00:00 |
+-------+------+---------------------+
5 rows in set (0.00 sec)
```