第一章 入门

一、入门简介

1、介绍

- MySQL是一种开放源代码的**关系型**数据库管理系统（RDBMS），使用最常用的数据库管理语言--**结构化查询语言**（SQL）进行数据库管理。
- MySQL是开放源代码的，因此任何人都可以在General Public License的许可下下载并根据个性化的需要对其进行修改。
- MySQL因为其速度、可靠性和适应性而备受关注。MySQL是管理内容最好的选择。

2、socket 介绍

- 客户端和服务端是使用socket链接的。
- socket中文翻译是插座，但是学术叫套接字。其实就是计算机和计算机链接的方式。
- 一个计算机使用ip+端口使用某种协议就能和另外一台机子的ip+端口链接通信。（TCP / UDP）
- mysql使用的默认端口是3306，一个客户端随便使用一个端口，就能连接上服务。
- 服务器的ip+端口是固定的,客户端往往是随机的。

二、SQL

1.SQL语句分类

- 1.DDL(Data Definition Language):数据定义语言，用来定义数据库对象：库、表、列等。功能：创建、删除、修改库和表结构。
- 2.DML(Data Manipulation Language):数据操作语言，用来定义数据库记录:增、删、改表记录。
- 3.DCL(Data Control Language):数据控制语言，用来定义访问权限和安全级别。
- 4.DQL(Data Query Language):数据查询语言，用来查询记录。也是本章学习的重点。

2、DCL(数据控制语言)语法（不重要）

该语言用来定义访问权限，理解即可，以后不会多用。

需要记住的是，一个项目创建一个用户，一个项目对应的数据库只有一个。这个用户只能对这个数据库有权限，其它数据库该用户就操作不了。

2.1 创建用户

用户只能在指定ip地址上登录mysql:`create user 用户名@IP地址 identified by '密码';`

用户可以在任意ip地址上登录:`create user 用户名@'%' identified by '密码';`

```
create user zn@'%' identified by 'zn123';
```

2.2 给用户授权

语法:`grant 权限1，…，权限n on 数据库.* to 用户名@IP地址；`其中权限1、2、n可以直接用all关键字代替。权限例如:create,alter,drop,insert,update,delete,select。

```
grant all on test.* to zn;
```

2.3 撤销授权

语法:`revoke 权限1，…，权限n on 数据库.* from 用户名@ ip地址；`撤销指定用户在指定数据库上的指定权限。撤销例如:`revoke create,delete on mydb1.* form user@localhost;`表示的意思是撤消user用户在数据库mydb1伤的create、alter权限。

```
revoke all on test.* from zn;
```

2.4 查看权限

查看指定用户的权限:`show grants for 用户名@ip地址；`

```
show grants for zn;
```

2.5 删除用户

```
drop user 用户名@ip地址；
```

3、DDL(数据定义语言)语法

该语言用来对数据库和表结构进行操作。

1、常用的数据类型

日期和时间数据类型

MySQL数据类型	含义
date	3字节，日期，格式：2014-09-18
time	3字节，时间，格式：08:42:30
datetime	8字节，日期时间，格式：2014-09-18 08:42:30
timestamp	4字节，自动存储记录修改的时间
year	1字节，年份

整型

MySQL数据类型	含义（有符号）
tinyint	1字节，范围（-128~127）
smallint	2字节，范围（-32768~32767）
mediumint	3字节，范围（-8388608~8388607）
int	4字节，范围（-2147483648~2147483647）
bigint	8字节，范围（+-9.22*10的18次方）

上面定义的都是有符号的，当然了，也可以加上unsigned关键字，定义成无符号的类型，那么对应的取值范围就要翻翻了，比如：

tinyint unsigned的取值范围为0~255。

浮点型

MySQL数据类型	含义
float(m, d)	4字节，单精度浮点型，m总个数，d小数位
double(m, d)	8字节，双精度浮点型，m总个数，d小数位
decimal(m, d)	decimal是存储为字符串的浮点数

我在MySQL中建立了一个表，有一列为float(5, 3)；做了以下试验：

1.插入123.45678，最后查询得到的结果为99.999；
2.插入123.456，最后查询结果为99.999；
3.插入12.34567，最后查询结果为12.346；

所以，在使用浮点型的时候，还是要注意陷阱的，要以插入数据库中的实际结果为准。

字符串数据类型

MySQL数据类型	含义
char(n)	固定长度，最多255个字符
varchar(n)	可变长度，最多65535个字符
tinytext	可变长度，最多255个字符
text	可变长度，最多65535个字符
mediumtext	可变长度，最多2的24次方-1个字符
longtext	可变长度，最多2的32次方-1个字符

1.char（n）和varchar（n）中括号中n代表字符的个数，并不代表字节个数，所以当使用了中文的时候(UTF8)意味着可以插入m个中文，但是实际会占用m*3个字节。
2.同时char和varchar最大的区别就在于char不管实际value都会占用n个字符的空间，而varchar只会占用实际字符应该占用的空间+1，并且实际空间+1<=n。
3.超过char和varchar的n设置后，字符串会被截断。
4.char的上限为255字节，varchar的上限65535字节，text的上限为65535。
5.char在存储的时候会截断尾部的空格，varchar和text不会。
6.varchar会使用1-3个字节来存储长度，text不会。

其它类型

1.enum(“member1", “member2", ... “member65535")
enum数据类型就是定义了一种枚举，最多包含65535个不同的成员。当定义了一个enum的列时，该列的值限制为列定义中声明的值。如果列声明包含NULL属性，则NULL将被认为是一个有效值，并且是默认值。如果声明了NOT NULL，则列表的第一个成员是默认值。

2.set(“member”, “member2", ... “member64")
set数据类型为指定一组预定义值中的零个或多个值提供了一种方法，这组值最多包括64个成员。值的选择限制为列定义中声明的值。

对数据库的操作（增删查改）：

查看所有数据库:`SHOW DATABASES;`

使用数据库:`USE 数据库名;`

创建数据库并指定编码，如不指定编码可能会有乱码问题，比如汉字不能存，当然建表指定也行，但是建库时指定一劳永逸:

```
CREATE DATABASE test DEFAULT CHARACTER SET utf8
```

删除数据库:`DROP DATABASE 数据库名;`

use test；切换数据库

对表结构的操作
创建表:

```
create table 表名 (

字段名1 类型 (宽度) 约束条件,
```

```
字段名2 类型(宽度) 约束条件,

字段名3 类型(宽度) 约束条件,

.......

);


CREATE TABLE student (
    id INT(10)  primary key,
    name VARCHAR (10),
    age INT (10) NOT NULL,
    gander varchar(2)
);

CREATE TABLE course (
  id INT (10)  primary key,
  name VARCHAR (10) ,
  t_id INT (10)
) ;

CREATE TABLE `teacher` (
  id INT (10)  primary key,
  name VARCHAR (10)
);

CREATE TABLE `scores` (
  s_id INT  primary key,
  score INT (10),
  c_id INT (10)
) ;
```

查看当前数据库中所有表:`SHOW TABLES;`

查看表结构:`DESC 表名;`

删除表:`DROP table 表名;`

修改表有5个操作，但前缀都是一样的:`ALTER TABLE 表名...` (不重要)

- 修改表之添加列:`ALTER TABLE 表名 add (列名 列类型, ..., 列名 列类型);`

  ```
  ALTER TABLE student add (address VARCHAR(20),hobby VARCHAR(20))
  ```

- 修改表之修改列类型:`ALTER TABLE 表名 MODIFY 列名 列的新类型;`

  ```
  ALTER TABLE student MODIFY hobby int;
  ```

- 修改表之列名称列类型一起修改:`ALTER TABLE 表名 CHANGE 原列名 新列名 列名类型;`

```
ALTER TABLE student CHANGE hobby newHobby VARCHAR(15);
```

- 修改表之删除列:`ALTER TABLE 表名 DROP 列名;`

```
ALTER TABLE student drop newHobby;
```

- 修改表之修改表名:`ALTER TABLE 表名 RENAME TO 新表名`

```
ALTER TABLE student RENAME to stu;
```

4、DML(数据操作语言)语法（重要）

该语言用来对表记录操作(增、删、改)。

4.1 插入数据(一次插入就是插入一行)

```
insert into 表名 (列名1，列名2，列名3) values (列值1，列值2，列值3);
insert into  stu (id,name,age,gander) values (2,'李华',19,'男');
```

说明:

1. 在数据库中所有的**字符串类型，必须使用单引号。**
2. (列名1，列名2，列名3)可省略，表示按照表中的顺序插入。但不建议采取这种写法，因为降低了程序的可读性。

4.2 修改记录

修改某列的全部值:`update 表名 set 列名1=列值1(，列名2=列值2);`

```
UPDATE stu set age=22;
UPDATE stu set age=23,name='张楠';
```

这样就全修改了，所以往往就要加条件。

```
UPDATE stu set age=23,name='张楠' where id = 1;
```

修改(某行或者多行记录的)列的指定值:`update 表名 set 列名1=列值1 where 列名2=列值2 or 列名3=列值3;`

运算符:`=、!=、<>、<、>、>=、<=、between...and、in(…)、is null、not、or、and`，其中in(...)的用法表示集合。例如:`update 表名 set 列名1=列值1 where 列名2=列值2 or 列名2=列值22`用in(...)写成`update 表名 set 列名1=列值1 where 列名2 in(列值2，列值3)`

4.4 删除数据(删除整行)

`delete from 表名 (where 条件);`不加where条件时会删除表中所有的记录，所以为了防止这种失误操作，很多数据库往往都会有备份。

第二章 建表约束

一、MySQL约束类型

约束名称	描述
NOT NULL	非空约束
UNIQUE	唯一约束，取值不允许重复,
PRIMARY KEY	主键约束（主关键字），自带非空、唯一、索引
FOREIGN KEY	外键约束（外关键字）
DEFAULT	默认值（缺省值）

二、MySQL约束类型举例

1. [NOT] NULL约束

```
drop table if EXISTS author;
CREATE TABLE author(
    aut_id varchar(8) NOT NULL,
    aut_name varchar(50) NOT NULL,
    country varchar(25) NOT NULL,
    home_city varchar(25) NOT NULL
);
```

2. UNIQUE约束

- **实现方法1（表的定义最后施加）**

```
drop table if EXISTS author;
CREATE TABLE IF NOT EXISTS author(
    aut_id varchar(8) NOT NULL ,
    aut_name varchar(50) NOT NULL,
    country varchar(25) NOT NULL,
    UNIQUE (aut_id)
);
```

- **实现方法2（字段定义的最后施加）**

```
drop table if EXISTS author;
CREATE TABLE IF NOT EXISTS author(
    aut_id varchar(8) NOT NULL UNIQUE ,
    aut_name varchar(50) NOT NULL,
    country varchar(25) NOT NULL
);
```

3. DEFAULT约束

```
drop table if EXISTS author;
CREATE TABLE IF NOT EXISTS author(
    aut_id varchar(8) NOT NULL UNIQUE ,
    aut_name varchar(50) NOT NULL,
    country varchar(25)  DEFAULT '中国'
);
```

5. PRIMARY KEY约束

> 主键只能有一个

> 但是可以由多个字段构成联合主键

- 单个字段作为主键（方法1）

```
drop table if EXISTS author;
CREATE TABLE author(
    aut_id int  PRIMARY KEY,
    aut_name varchar(50) NOT NULL,
    country varchar(25) NOT NULL
);
```

- 单个字段作为主键（方法2）

```
drop table if EXISTS author;
CREATE TABLE author(
    aut_id varchar(8),
    aut_name varchar(50) NOT NULL,
    country varchar(25) NOT NULL,
    PRIMARY KEY (aut_id)
);
```

- 多个字段作为主键

```
drop table if EXISTS author;
CREATE TABLE author(
    aut_id varchar(8)  not null,
    aut_name varchar(50) NOT NULL,
    country varchar(25) NOT NULL,
    PRIMARY KEY (aut_id, country)
);
```

###

6. AUTO_INCREMENT约束

> 需要配合主键使用

```
drop table if EXISTS author;
CREATE TABLE author(
    aut_id varchar(8)  PRIMARY KEY AUTO_INCREMENT,
    aut_name varchar(50) NOT NULL,
    country varchar(25) NOT NULL
);
```

7. FOREIGN KEY约束

> 外键可以建立多个，多个外键接着写就行了
>
> 外键会产生的效果
>
> 1、删除表时，如果不删除引用外键的表，被引用的表不能直接删除
>
> 2、外键的值必须来源于引用的表的主键字段

语法：

> FOREIGN KEY [column list] REFERENCES [primary key table] ([column list]);

```
drop table if EXISTS author;
CREATE table author(
    aut_id int PRIMARY key auto_increment,
    aut_name VARCHAR(10) not null,
    country VARCHAR(20) DEFAULT '中国'
);

drop table if EXISTS book;
CREATE TABLE IF NOT EXISTS book(
    book_id int PRIMARY key auto_increment,
    book_name varchar(50)  ,
    aut_id int ,
    book_price decimal(8,2) ,
    FOREIGN KEY (aut_id) REFERENCES author(aut_id)
);


INSERT into author (aut_name) VALUES('张妈');
INSERT into book (book_name,aut_id) VALUES('王阳明大传2',2);
```

第三章 查询语言

重点，该语言用来查询记录，不会修改数据库和表结构。

```
insert into student (id,name,age,gander) values (1,'zhangsan',13,'男');
insert into student (id,name,age) values (2,'lisi',13);
insert into student values (3,'wangwu',13,'男');

update student set age = 25;
UPDATE student set age = 13 where id = 2;
UPDATE student set age = 15,gander = '女' where id > 1;
UPDATE student set age = 90 where id < 3 and  gander='女';

DELETE from student where age > 15;
DELETE from student where id = 3;
DELETE FROM student;
```

一、构建数据库

创建数据库以及创建表单:

```
drop TABLE if EXISTS student;
CREATE TABLE student (
    id INT(10) PRIMARY key,
    name VARCHAR (10),
    age INT (10) NOT NULL,
    gander varchar(2)
);

drop TABLE if EXISTS course;
CREATE TABLE course (
  id INT (10)  PRIMARY key,
  name VARCHAR (10) ,
  t_id INT (10)
) ;

drop TABLE if EXISTS teacher;
CREATE TABLE teacher(
  id INT (10)  PRIMARY key,
  name VARCHAR (10)
);

drop TABLE if EXISTS scores;
CREATE TABLE scores(
  s_id INT ,
  score INT (10),
  c_id INT (10) ,
    PRIMARY key(s_id,c_id)
```

```
) ;
```

表单填充数据：

```
insert into  student (id,name,age,gander)VALUES(1,'白杰',19,'男');
insert into  student (id,name,age,gander)VALUES(2,'连宇栋',19,'男');
insert into  student (id,name,age,gander)VALUES(3,'邸志伟',24,'男');
insert into  student (id,name,age,gander)VALUES(4,'李兴',11,'男');
insert into  student (id,name,age,gander)VALUES(5,'张琪',18,'男');
insert into  student (id,name,age,gander)VALUES(6,'武三水',18,'女');
insert into  student (id,name,age,gander)VALUES(7,'张志伟',16,'男');
insert into  student (id,name,age,gander)VALUES(8,'康永亮',23,'男');
insert into  student (id,name,age,gander)VALUES(9,'杨涛瑞',22,'女');
insert into  student (id,name,age,gander)VALUES(10,'王杰',21,'男');

insert into  course (id,name,t_id)VALUES(1,'数学',1);
insert into  course (id,name,t_id)VALUES(2,'语文',2);
insert into  course (id,name,t_id)VALUES(3,'c++',3);
insert into  course (id,name,t_id)VALUES(4,'java',4);
insert into  course (id,name)VALUES(5,'php');


insert into  teacher (id,name)VALUES(1,'张楠');
insert into  teacher (id,name)VALUES(2,'老孙');
insert into  teacher (id,name)VALUES(3,'薇薇姐');
insert into  teacher (id,name)VALUES(4,'磊磊哥');
insert into  teacher (id,name)VALUES(5,'大微姐');


insert into  scores (s_id,score,c_id)VALUES(1,80,1);
insert into  scores (s_id,score,c_id)VALUES(1,56,2);
insert into  scores (s_id,score,c_id)VALUES(1,95,3);
insert into  scores (s_id,score,c_id)VALUES(1,30,4);
insert into  scores (s_id,score,c_id)VALUES(1,76,5);

insert into  scores (s_id,score,c_id)VALUES(2,35,1);
insert into  scores (s_id,score,c_id)VALUES(2,86,2);
insert into  scores (s_id,score,c_id)VALUES(2,45,3);
insert into  scores (s_id,score,c_id)VALUES(2,94,4);
insert into  scores (s_id,score,c_id)VALUES(2,79,5);

insert into  scores (s_id,score,c_id)VALUES(3,65,2);
insert into  scores (s_id,score,c_id)VALUES(3,85,3);
insert into  scores (s_id,score,c_id)VALUES(3,37,4);
insert into  scores (s_id,score,c_id)VALUES(3,79,5);

insert into  scores (s_id,score,c_id)VALUES(4,66,1);
insert into  scores (s_id,score,c_id)VALUES(4,39,2);
insert into  scores (s_id,score,c_id)VALUES(4,85,3);

insert into  scores (s_id,score,c_id)VALUES(5,66,2);
insert into  scores (s_id,score,c_id)VALUES(5,89,3);
insert into  scores (s_id,score,c_id)VALUES(5,74,4);


insert into  scores (s_id,score,c_id)VALUES(6,80,1);
insert into  scores (s_id,score,c_id)VALUES(6,56,2);
```

```
insert into  scores (s_id,score,c_id)VALUES(6,95,3);
insert into  scores (s_id,score,c_id)VALUES(6,30,4);
insert into  scores (s_id,score,c_id)VALUES(6,76,5);

insert into  scores (s_id,score,c_id)VALUES(7,35,1);
insert into  scores (s_id,score,c_id)VALUES(7,86,2);
insert into  scores (s_id,score,c_id)VALUES(7,45,3);
insert into  scores (s_id,score,c_id)VALUES(7,94,4);
insert into  scores (s_id,score,c_id)VALUES(7,79,5);

insert into  scores (s_id,score,c_id)VALUES(8,65,2);
insert into  scores (s_id,score,c_id)VALUES(8,85,3);
insert into  scores (s_id,score,c_id)VALUES(8,37,4);
insert into  scores (s_id,score,c_id)VALUES(8,79,5);

insert into  scores (s_id,score,c_id)VALUES(9,66,1);
insert into  scores (s_id,score,c_id)VALUES(9,39,2);
insert into  scores (s_id,score,c_id)VALUES(9,85,3);
insert into  scores (s_id,score,c_id)VALUES(9,79,5);

insert into  scores (s_id,score,c_id)VALUES(10,66,2);
insert into  scores (s_id,score,c_id)VALUES(10,89,3);
insert into  scores (s_id,score,c_id)VALUES(10,74,4);
insert into  scores (s_id,score,c_id)VALUES(10,79,5);
```

二、单表查询

1、基本查询(后缀都是统一为from 表名)

(1) 查询所有列:`select * from 表名;`其中 * 表示查询所有列，而不是所有行的意思。

(2) 查询指定列:`select 列1，列2，列n from 表名;`

(3) 完全重复的记录只显示一次:在查询的列之前添加distinct

(4) 列运算

```
a.数量类型的列可以做加、减、乘、除:`SELECT sal*5 from 表名；`说明:1.遇到null加任何值都等于null的情况，需要用到ifnull()函数。2.将字符串做加减乘除运算，会把字符串当作0。

b.字符串累类型可以做连续运算(需要用到concat()函数):`select concat(列名1，列名2) from 表名;`其中列名的类型要为字符串。

c. 给列名起别名:`select 列名1 (as) 别名1,列名2 (as) 别名2 from 表名；`
```

(5) 条件控制

```
a.条件查询。在后面添加where指定条件:`select * from 表名 where 列名=指定值;`
b.模糊查询：当你想查询所有姓张的记录。用到关键字like。

select * from 表名 where 列名 like '张_';
(_代表匹配任意一个字符，%代表匹配0～n个任意字符)。
```

2、排序(所谓升序和降序都是从上往下排列)

- 1.升序:`select * form 表名 order by 列名 (ASC );` ()里面的内容为缺省值;
- 2.降序:`select * from 表名 order by 列名 DESC;`
- 3.使用多列作为排序条件: 当第一列排序条件相同时，根据第二列排序条件排序(当第二列依旧相同时可视情况根据第三例条件排序)。eg:`select * from 表名 order by 列名1 ASC, 列名2 DESC;` 意思是当列名1的值相同时按照列名2的值降序排。

3、聚合函数

- 1.count:`select count(列名) from 表名;`,纪录行数。
- 2.max:`select max(列名) from 表名;`,列中最大值。
- 3.min:`select min(列名) from 表名;`,列中最小值。
- 4.sum:`select sum(列名) from 表名;`,求列的总值，null 和字符串默认为0。
- 5.avg:`select avg(列名) from 表名;`,一列的平均值。

4、分组查询

分组查询的信息都是组的信息，不能查到个人的信息，其中查询组的信息是通过聚合函数得到的。

语法：`select 分组列名，聚合函数1,聚合函数2 from 表名 group by 该分组列名;`其中分组列名需要的条件是该列名中有重复的信息。

查询的结果只能为：作为分组条件的列和聚合函数；查处的信息都是组的信息。

分组查询前，还可以通过关键字where先把满足条件的人分出来，再分组。语法为:`select 分组列，聚合函数 from 表名 where 条件 group by 分组列;`

分组查询后，也可以通过关键字having把组信息中满足条件的组再细分出来。语法为：`select 分组列，聚合函数 from 表名 where 条件 group by 分组列 having 聚合函数或列名(条件);`

```
select gander,avg(age) avg_age,sum(age) sum_age from student GROUP BY gander
HAVING  gander = '男'
```

5、LIMIT子句(mysql中独有的语法)

LIMIT用来限定查询结果的起始行，以及总行数。

例如：`select * from 表名 limit 4，3;`表示起始行为第5行，一共查询3行记录。

```
--如果一个参数 说明从开始查找三条记录
SELECT id,name,age,gander FROM student limit 3
--如果两个参数 说明从第三行起（不算），向后查三条记录
SELECT id,name,age,gander FROM student limit 3,3
```

union

union all

三、多表查询

笛卡尔积：简单来说就是两个集合相乘的结果，集合A和集合B中**任意**两个元素结合在一起。

1、内连接

内连接

内连接查询操作只列出与连接条件匹配的数据行，使用**INNER JOIN**或者直接使用**JOIN** 进行连接。

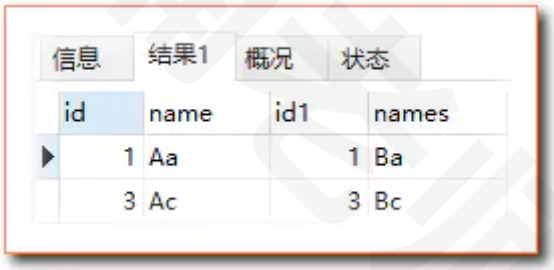
```
SELECT * from Table_A JOIN Table_B;
SELECT * from Table_A INNER JOIN Table_B;
```

内连接可以没有连接条件，没有条件之后的查询结果，会保留所有结果（**笛卡尔集**），与后面分享的交叉连接差不多。

在连接条件中使用**等于号（=）**运算符比较被连接列的列值，其查询结果中列出被连接表中的所有列，**包括其中的重复列。**

```
SELECT * from Table_A A JOIN Table_B B ON A.id = B.id;
```

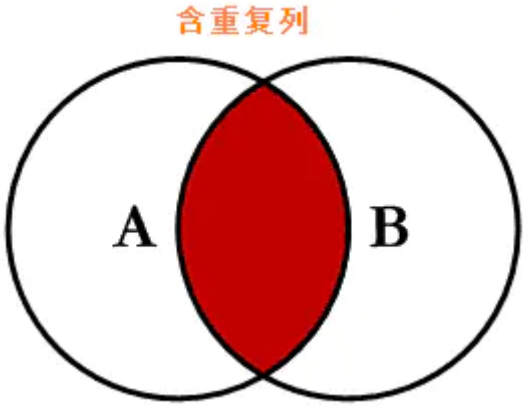
查询结果，注意列数是 4 列，两张表的字段直接拼接在一起，重复的字段在后面添加数字序列以做区分



信息 | 结果1 | 概况 | 状态

id	name	id1	names
1	Aa	1	Ba
3	Ac	3	Bc

通俗讲就是根据条件，找到表 A 和 表 B 的数据的交集



例子：

```
普通的多表查，课内连接接通相同
SELECT * from teacher t , course c where t.id = c.t_id
（这样会先生成笛卡尔积，效率可能略低）

SELECT * from teacher t JOIN course c on t.id = c.t_id
SELECT * from teacher t inner JOIN course c on t.id = c.t_id

结果：只有满足条件的会显示，5号老师没课程，5号课程没老师都不会显示
1    王宝强 1    数学   1
2    贾宝玉 2    语文   2
3    温迪   3    c++ 3
4    路人甲 4    java     4
```

2、外连接（常用）

外连接**不只**列出与连接条件相匹配的行，而且还加上左表(左外连接时)或右表(右外连接时)或两个表(全外连接时)中所有符合搜索条件的数据行。

（1）左连接（左外连接）

```sql
SELECT * from Table_A A LEFT JOIN Table_B B ON A.id = B.id;
SELECT * from Table_A A LEFT OUTER JOIN Table_B B ON A.id = B.id;
```

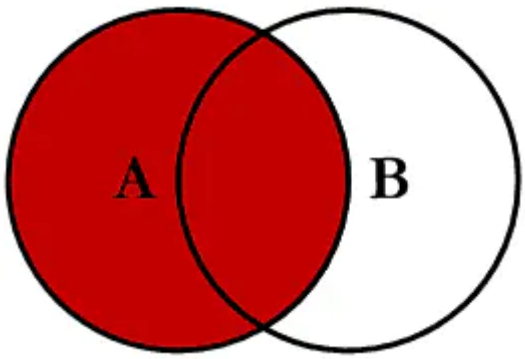
查询结果如下

信息 | 结果1 | 概况 | 状态

id	name	id1	names
1	Aa	1	Ba
3	Ac	3	Bc
4	Ad	(Null)	(Null)
8	Ae	(Null)	(Null)

根据条件，用右表（B）匹配左表（A），能匹配，正确保留，**不能匹配其他表的字段都置空 Null。**

也就是，根据条件找到表 A 和 表 B 的数据的交集，再加上左表的数据集， Venn 图表示就是



红色部分代表查询结果

例子：

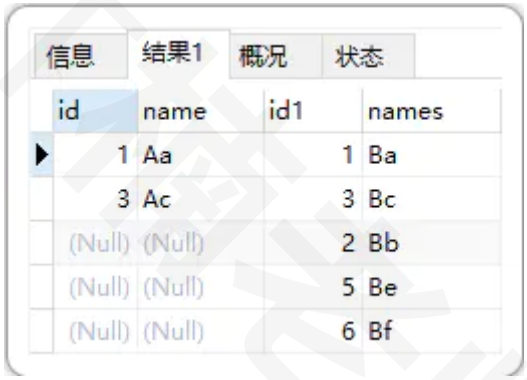
```
SELECT * from teacher t LEFT JOIN course c on t.id = c.t_id
```

```
结果：只有满足条件的会显示，5号老师没课程，依然显示，5号课程没老师都不会显示，
左边表的所有数据都显示

1    王宝强 1    数学   1
2    贾宝玉 2    语文   2
3    温迪   3    c++ 3
4    路人甲 4    java     4
5    路人乙
```

（2）右连接（右外连接）

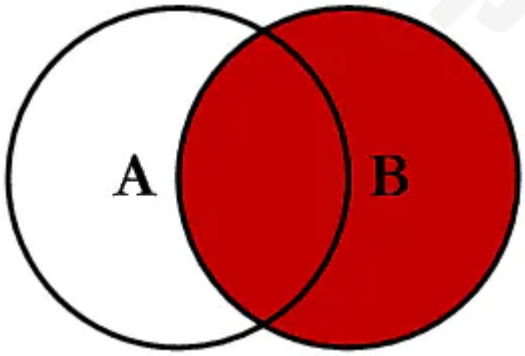
```
1   SELECT * from Table_A A RIGHT JOIN Table_B B ON A.id=B.id;
2   SELECT * from Table_A A RIGHT OUTER JOIN Table_B B ON A.id=B.id;
```

查询结果如下



根据条件，用左表（A）匹配右表（B），能匹配，正确保留，**不能匹配其他表的字段都置空 Null。**

也就是，根据条件找到表 A 和 表 B 的数据的交集，再加上右表的数据集， Venn 图表示就是



例子：

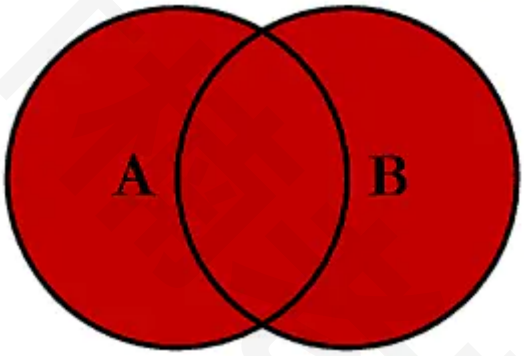
```
SELECT * from teacher t right JOIN course c on t.id = c.t_id

结果：只有满足条件的会显示，5号老师没课程不显示，5号课程没老师都，依然显示，
右边表的所有数据都显示

1   王宝强 1   数学  1
2   贾宝玉 2   语文  2
3   温迪  3   c++ 3
4   路人甲 4   java    4
          5    php
```

3、全连接，mysql不支持，oracle支持

```
SELECT * from Table_A A FULL JOIN Table_B B ON A.id=B.id;
SELECT * from Table_A A FULL OUTER JOIN Table_B B ON A.id=B.id;
```

目前我的 MySQL 不支持此种方式，可以用其他方式替代解决，在此不展开。

理论上是根据条件找到表 A 和 表 B 的数据的交集，再加上左右表的数据集



四、子查询

1、where 型子查询

将查询结果当条件

例子：查询有**一门学科分数大于八十分**的**学生信息**

```
SELECT * from student where id in
(select DISTINCT s_id from scores where score > 90);
```

where 型子查询，如果是 where 列 =（内层 sql） 则内层 sql 返回的必须是单行单列，单个值。

where 型子查询，如果是 where 列 in（内层 sql） 则内层 sql 返回的必须是单列，可以多行。

2、from 型子查询

在学习 from 子查询之前，需要理解一个概念：**查询结果集在结构上可以当成表看**，那就可以当成临时表对他进行再次查询：

取排名**数学成绩**前五名的学生，正序排列。

```
select * from  (SELECT s.id,s.name,e.score,c.`name` cname from student s LEFT
JOIN scores e
on s.id = e.s_id left JOIN course c on e.c_id = c.id
where c.`name` = '数学' order by e.score desc limit 5 ) t ORDER BY t.score
```

五、练习题

1、查询'01'号学生的姓名和各科成绩。 *

```
select
s.`name` "学生名字",
e.name "课程名称",
c.score "成绩"
from student s
LEFT JOIN scores c
on s.id = c.s_id
LEFT JOIN course e
on e.id = c.c_id
where s.id = 1
```

2、查询各个学科的平均成绩，最高成绩。 *

3、查询每个同学的最高成绩及科目名称。 *

4、查询所有姓张的同学的各科成绩。 *

5、查询每个课程最高分的同学信息。 *

6、查询名字中含有"张"和'李'字的学生信息和各科成绩 *

7、查询平均成绩及格的同学的信息。*

8、将学生按照总分数进行排名。 *

9、查询数学成绩的最高分、最低分、平均分。 **

10、将各科目按照平均分排序。 **

11、查询老师的信息和他所带科目的平均分。 **

12、查询被"张楠"和"老孙"叫的课程的最高分和平均分。 **

13、查询查询每个同学的最好成绩的科目名称。 **

14、查询所有学生的课程及分数。**

15、查询课程编号为01且课程成绩在80分以上的学生的学号和姓名。**

```
select s.id,s.name from student s where
s.id in (
    select c.s_id from  scores c
    where c.c_id = '1' and c.score > 40
)
```

16、查询平均成绩大于等于85的所有学生的学号、姓名和平均成绩。 **

17、查询有不及格课程的同学信息。 **

18、求每门课程的学生人数。 **

19、查询每门课程的平均成绩，结果按平均成绩降序排列，平均成绩相同时，按课程编号升序排列 。 ***

20、查询平均成绩大于等于60分的同学的学生编号和学生姓名和平均成绩。 ***

21、查询有一门课程成绩在90分以上的学生信息； ***

22、查询出只有三门课程的全部学生的学号和姓名 ***

23、查询有不及格课程的课程信息 ***

24、检索至少选修四门课程的学生学号 ***

25、查询没有学全所有课程的同学的信息 ***

26、查询学全所有课程的同学的信息。 ****

27、 查询各学生都选了多少门课 ***

28、查询课程名称为"java"，且分数低于60的学生姓名和分数。 ***

```
select s.`name`,m.score  from student s left join
(select c.s_id,c.score
from  scores c left JOIN course t
on c.c_id = t.id
where t.name = 'java' and c.score < 60) m
on m.s_id = s.id
where m.s_id is not null
```

29、查询学过"张楠"老师授课的同学的信息 。 ****

30、查询没学过"张楠"老师授课的同学的信息 ****

第四章 MySQL常用函数介绍

MySQL数据库中提供了很丰富的函数，比如我们常用的聚合函数，日期及字符串处理函数等。SELECT语句及其条件表达式都可以使用这些函数，函数可以帮助用户更加方便的处理表中的数据，使MySQL数据库的功能更加强大。本篇文章主要为大家介绍几类常用函数的用法。

1.聚合函数

聚合函数是平时比较常用的一类函数，这里列举如下：

- COUNT(col) 统计查询结果的行数
- MIN(col) 查询指定列的最小值
- MAX(col) 查询指定列的最大值
- SUM(col) 求和，返回指定列的总和
- AVG(col) 求平均值，返回指定列数据的平均值

2.数值型函数

数值型函数主要是对数值型数据进行处理，得到我们想要的结果，常用的几个列举如下，具体使用方法大家可以试试看。

- CEILING(x) 返回大于x的最小整数值，向上取整
- FLOOR(x) 返回小于x的最大整数值，向下取整
- ROUND(x,y) 返回参数x的四舍五入的有y位小数的值 四舍五入
- TRUNCATE(x,y) 返回数字x截短为y位小数的结果
- PI() 返回pi的值（圆周率）
- RAND() 返回 0 到 1 内的随机值,可以通过提供一个参数(种子)使RAND()随机数生成器生成一个指定的值

一些示例：

```
# ABS()函数求绝对值
mysql> SELECT ABS(5),ABS(-2.4),ABS(-24),ABS(0);
+--------+-----------+----------+--------+
| ABS(5) | ABS(-2.4) | ABS(-24) | ABS(0) |
+--------+-----------+----------+--------+
|      5 |       2.4 |       24 |      0 |
+--------+-----------+----------+--------+

# 取整函数 CEIL(x) 和 CEILING(x) 的意义相同，返回不小于 x 的最小整数值
mysql> SELECT CEIL(-2.5),CEILING(2.5);
+------------+--------------+
| CEIL(-2.5) | CEILING(2.5) |
+------------+--------------+
|         -2 |            3 |
+------------+--------------+

# 求余函数 MOD(x,y) 返回 x 被 y 除后的余数
mysql> SELECT MOD(63,8),MOD(120,10),MOD(15.5,3);
+-----------+-------------+-------------+
| MOD(63,8) | MOD(120,10) | MOD(15.5,3) |
+-----------+-------------+-------------+
|         7 |           0 |         0.5 |
+-----------+-------------+-------------+

# RAND() 函数被调用时，可以产生一个在 0 和 1 之间的随机数
mysql> SELECT RAND(), RAND(), RAND();
+--------------------+--------------------+---------------------+
| RAND()             | RAND()             | RAND()              |
+--------------------+--------------------+---------------------+
| 0.24996517063115273 | 0.9559759106077029 | 0.029984071878701515 |
+--------------------+--------------------+---------------------+
```

3.字符串函数

字符串函数可以对字符串类型数据进行处理，在程序应用中用处还是比较大的，同样这里列举几个常用的如下：

- LENGTH(s) 计算字符串长度函数，返回字符串的字节长度
- CONCAT(s1,s2...,sn) 合并字符串函数，返回结果为连接参数产生的字符串，参数可以是一个或多个
- LOWER(str) 将字符串中的字母转换为小写
- UPPER(str) 将字符串中的字母转换为大写

- LEFT(str,x) 返回字符串str中最左边的x个字符
- RIGHT(str,x) 返回字符串str中最右边的x个字符
- TRIM(str) 删除字符串左右两侧的空格
- REPLACE 字符串替换函数，返回替换后的新字符串 REPLACE(name,'白','黑')
- SUBSTRING 截取字符串，返回从指定位置开始的指定长度的字符换
- REVERSE(str) 返回颠倒字符串str的结果

一些示例：

```
# LENGTH(str) 函数的返回值为字符串的字节长度
mysql> SELECT LENGTH('name'),LENGTH('数据库');
+---------------+--------------------+
| LENGTH('name') | LENGTH('数据库')    |
+---------------+--------------------+
|              4 |                  9 |
+----------------+--------------------+

# CONCAT(s1，s2，...) 函数返回结果为连接参数产生的字符串 若有任何一个参数为 NULL，则返回值为
NULL
mysql> SELECT CONCAT('MySQL','5.7'),CONCAT('MySQL',NULL);
+----------------------+----------------------+
| CONCAT('MySQL','5.7') | CONCAT('MySQL',NULL) |
+----------------------+----------------------+
| MySQL5.7              | NULL                 |
+----------------------+----------------------+

# INSERT(s1，x，len，s2) 返回字符串 s1，子字符串起始于 x 位置，并且用 len 个字符长的字符串
代替 s2
mysql> SELECT INSERT('Football',2,4,'Play') AS col1,
    -> INSERT('Football',-1,4,'Play') AS col2,
    -> INSERT('Football',3,20,'Play') AS col3;
+----------+----------+--------+
| col1     | col2     | col3   |
+----------+----------+--------+
| FPlayall | Football | FoPlay |
+----------+----------+--------+

# UPPER,LOWER是大小写转换函数
mysql> SELECT LOWER('BLUE'),LOWER('Blue'),UPPER('green'),UPPER('Green');
+---------------+---------------+----------------+----------------+
| LOWER('BLUE') | LOWER('Blue') | UPPER('green') | UPPER('Green') |
+---------------+---------------+----------------+----------------+
| blue          | blue          | GREEN          | GREEN          |
+---------------+---------------+----------------+----------------+

# LEFT,RIGHT是截取左边或右边字符串函数
mysql> SELECT LEFT('MySQL',2),RIGHT('MySQL',3);
+-----------------+-----------------+
| LEFT('MySQL',2) | RIGHT('MySQL',3) |
+-----------------+-----------------+
| My              | SQL             |
+-----------------+-----------------+

# REPLACE(s，s1，s2) 使用字符串 s2 替换字符串 s 中所有的字符串 s1
mysql> SELECT REPLACE('aaa.mysql.com','a','w');
```

```
+----------------------------------+
| REPLACE('aaa.mysql.com','a','w') |
+----------------------------------+
| www.mysql.com                    |
+----------------------------------+

# 函数 SUBSTRING(s，n，len) 带有 len 参数的格式，从字符串 s 返回一个长度同 len 字符相同的子字符串，起始于位置 n
mysql> SELECT SUBSTRING('computer',3) AS col1,
    -> SUBSTRING('computer',3,4) AS col2,
    -> SUBSTRING('computer',-3) AS col3,
    -> SUBSTRING('computer',-5,3) AS col4;
+--------+------+------+------+
| col1   | col2 | col3 | col4 |
+--------+------+------+------+
| mputer | mput | ter  | put  |
+--------+------+------+------+
```

4.日期和时间函数

- CURDATE 和 CURRENT_DATE 两个函数作用相同，返回当前系统的日期值
- CURTIME 和 CURRENT_TIME 两个函数作用相同，返回当前系统的时间值
- **NOW** 和 SYSDATE 两个函数作用相同，返回当前系统的日期和时间值

- UNIX_TIMESTAMP 获取UNIX时间戳函数，返回一个以 UNIX 时间戳为基础的无符号整数
- **FROM_UNIXTIME 将 UNIX 时间戳转换为时间格式，与UNIX_TIMESTAMP互为反函数**

- MONTH 获取指定日期中的月份
- MONTHNAME 获取指定日期中的月份英文名称
- DAYNAME 获取指定日期对应的星期几的英文名称
- DAYOFWEEK 获取指定日期对应的一周的索引位置值
- WEEK 获取指定日期是一年中的第几周，返回值的范围是否为 0∽52 或 1∽53
- DAYOFYEAR 获取指定日期是一年中的第几天，返回值范围是1~366
- DAYOFMONTH 获取指定日期是一个月中是第几天，返回值范围是1~31
- YEAR 获取年份，返回值范围是 1970∽2069

- DATE_ADD 和 ADDDATE 两个函数功能相同，都是向日期添加指定的时间间隔
- DATE_SUB 和 SUBDATE 两个函数功能相同，都是向日期减去指定的时间间隔
- ADDTIME 时间加法运算，在原始时间上添加指定的时间
- SUBTIME 时间减法运算，在原始时间上减去指定的时间
- DATEDIFF 获取两个日期之间间隔，返回参数 1 减去参数 2 的值
- **DATE_FORMAT 格式化指定的日期，根据参数返回指定格式的值**

当使用了表达式计算后，不能直接使用别名进行判断了。

一些示例：

```
# CURDATE() 和 CURRENT_DATE() 函数的作用相同，将当前日期按照“YYYY-MM-DD”或“YYYYMMDD”格式的值返回
mysql> SELECT CURDATE(),CURRENT_DATE(),CURRENT_DATE()+0;
+------------+----------------+------------------+
| CURDATE()  | CURRENT_DATE() | CURRENT_DATE()+0 |
+------------+----------------+------------------+
| 2019-10-22 | 2019-10-22     |         20191022 |
+------------+----------------+------------------+

# MONTH(date) 函数返回指定 date 对应的月份
mysql> SELECT MONTH('2017-12-15');
+---------------------+
| MONTH('2017-12-15') |
+---------------------+
|                  12 |
+---------------------+

# DATE_ADD(date,INTERVAL expr type) 和 ADDDATE(date,INTERVAL expr type) 两个函数的作用相同，都是用于执行日期的加运算。
mysql> SELECT DATE_ADD('2018-10-31 23:59:59',INTERVAL 1 SECOND) AS C1,
    -> DATE_ADD('2018-10-31 23:59:59',INTERVAL '1:1' MINUTE_SECOND) AS C2,
    -> ADDDATE('2018-10-31 23:59:59',INTERVAL 1 SECOND) AS C3;
+---------------------+---------------------+---------------------+
| C1                  | C2                  | C3                  |
+---------------------+---------------------+---------------------+
| 2018-11-01 00:00:00 | 2018-11-01 00:01:00 | 2018-11-01 00:00:00 |
+---------------------+---------------------+---------------------+

# DATEDIFF(date1，date2) 返回起始时间 date1 和结束时间 date2 之间的天数
mysql> SELECT DATEDIFF('2017-11-30','2017-11-29') AS COL1,
    -> DATEDIFF('2017-11-30','2017-12-15') AS col2;
+------+------+
| COL1 | col2 |
+------+------+
|    1 |  -15 |
+------+------+

# DATE_FORMAT(date，format) 函数是根据 format 指定的格式显示 date 值
mysql> SELECT DATE_FORMAT('2017-11-15 21:45:00','%W %M %D %Y') AS col1,
    -> DATE_FORMAT('2017-11-15 21:45:00','%h:i% %p %M %D %Y') AS col2;
+------------------------------+----------------------------+
| col1                         | col2                       |
+------------------------------+----------------------------+
| Wednesday November 15th 2017 | 09:i PM November 15th 2017 |
+------------------------------+----------------------------+
```

5.流程控制函数

流程控制类函数可以进行条件操作，用来实现SQL的条件逻辑，允许开发者将一些应用程序业务逻辑转换到数据库后台，列举如下：

- IF(test,t,f) 如果test是真，返回t；否则返回f
- IFNULL(arg1,arg2) 如果arg1不是空，返回arg1，否则返回arg2
- NULLIF(arg1,arg2) 如果arg1=arg2返回NULL；否则返回arg1

- CASE WHEN[test1] THEN [result1]...ELSE [default] END 如果testN是真，则返回resultN，否则返回default
- CASE [test] WHEN[val1] THEN [result]...ELSE [default]END 如果test和valN相等，则返回resultN，否则返回default
- CASE 列名
- WHEN condition THEN result

  WHEN condition THEN result

```
.............
[WHEN ...]
[ELSE result]
```

  END

语文　数学　英语

张三

列转行案例

```
CREATE TABLE `mystudent` (
  `ID` int(10) NOT NULL AUTO_INCREMENT,
  `USER_NAME` varchar(20) DEFAULT NULL,
  `COURSE` varchar(20) DEFAULT NULL,
  `SCORE` float DEFAULT '0',
  PRIMARY KEY (`ID`)
)

insert into mystudent(USER_NAME, COURSE, SCORE)  values
("张三", "数学", 34),
("张三", "语文", 58),
("张三", "英语", 58),
("李四", "数学", 45),
("李四", "语文", 87),
("李四", "英语", 45),
("王五", "数学", 76),
("王五", "语文", 34),
("王五", "英语", 89);

SELECT user_name ,
    MAX(CASE course WHEN '数学' THEN score ELSE 0 END ) 数学,
    MAX(CASE course WHEN '语文' THEN score ELSE 0 END ) 语文,
    MAX(CASE course WHEN '英语' THEN score ELSE 0 END ) 英语
FROM mystudent
GROUP BY USER_NAME;

select * from mystudent
```

6.加密函数

- **MD5()** 计算字符串str的MD5校验和

第五章 数据库设计

一、三范式

注：设计只是一种思想一种理念，我们按照规范的设计方式设计数据库对我们来说有好处，但绝对不是说一定要严格遵守，三范式能极大的**减少数据冗余，但是相对编写sql而言是增加了难度的，所以所有好的设计都是要权衡利弊的，要对编码难度，存储大小，执行效率等多方面进行综合考量，但是在学习初期最好紧紧的遵循三范式，在后续的编码中体会和总结自己的经验。

设计数据库表的时候所依据的规范，共三个规范：

```
第一范式：要求有主键，并且要求每一个字段原子性不可再分
第二范式：要求所有非主键字段完全依赖主键，不能产生部分依赖
第三范式：所有非主键字段和主键字段之间不能产生传递依赖
```

第一范式

数据库表中不能出现重复记录，每个字段是原子性的不能再分

不符合第一范式的实例：

学生编号	学生姓名	联系方式
1001	白杰	bj@qq.com,18565987896
1002	杨春旺	ycw@qq.com,13659874598
1003	张志伟	zzw@qq.com,12598745698

解决方案

学生编号	学生姓名	邮箱地址	联系电话
1001	白杰	bj@qq.com	18565987896
1002	杨春旺	ycw@qq.com	13659874598
1003	张志伟	zzw@qq.com	12598745698

不符合第一范式的实例，不是说他错哈：

学生编号	学生姓名	联系地址
1001	白杰	太原市尖草坪区恒山路108号
1002	杨春旺	太原市迎泽区迎泽大家100号
1003	张志伟	太原市杏花岭区北大街152号

解决方案：

学生编号	学生姓名	市	区	详细地址
1001	白杰	太原市	尖草坪区	恒山路108号
1002	杨春旺	太原市	迎泽区	迎泽大街100号
1003	张志伟	太原市	杏花岭区	北大街152号

必须有主键，这是数据库设计的最基本要求，主要采用数值型或定长字符串表示，**关于列不可再分，应该根据具体的情况来决定。**如联系方式，为了开发上的便利可能就采用一个字段。

关于第一范式，每一行必须唯一，也就是每个表必须有主键，这是数据库设计的最基本要求，主要采用数值型或定长字符串表示，**关于列不可再分，应该根据具体的情况来决定。**如联系方式，为了开发上的便利可能就采用一个字段。

第二范式

第二范式是建立在第一范式基础上的，另外要求所有非主键字段完全依赖主键，不能产生**部分依赖**

不符合第二范式的案例：

其中学生编号和课程编号为联合主键

学生编号	性别	学生姓名	课程编号	课程名称	教室	成绩
1001	男	白杰	2001	java	3004	89
1002	男	杨春旺	2002	mysql	3003	88
1003	女	刘慧慧	2003	html	3005	90
1001	男	白杰	2002	mysql	3003	77
1001	男	白杰	2003	html	3005	89
1003	女	刘慧慧	2001	java	3004	90

以上虽然确定了主键，但此表会出现大量的数据冗余，出现冗余的原因在于，学生信息部分依赖了主键的一个字段学生编号，和课程id没有毛线关系。同时课程的信息只是依赖课程id，和学生id没有毛线关系。只有成绩一个字段完全依赖主键的两个部分，这就是第二范式**部分依赖。**

解决方案：

学生表：学生编号为主键

学生编号	性别	学生姓名
1001	男	白杰
1002	男	杨春旺
1003	女	刘慧慧

课程表：课程编号为主键

课程编号	课程名称	教室
2001	java	3003
2002	mysql	3003
2003	html	3005

成绩表：学生编号和课程编号为联合主键

学生编号	课程编号	成绩
1001	2001	89
1002	2002	88
1003	2003	90
1001	2002	77
1001	2003	89
1003	2001	90

如果一个表是单一主键，那么它就是复合第二范式，部分依赖和主键有关系

以上是典型的“多对多”设计

第三范式

建立在第二范式基础上的，非主键字段不能传递依赖于主键字段（不要产生传递依赖）

不满足第三范式的例子：

其中学生编号是主键

学生编号	学生姓名	专业编号	专业名称
1001	白杰	2001	计算机
1002	杨春旺	2002	自动化
1003	张志伟	2001	计算机

何为传递依赖

专业编号依赖学生编号，应为该学生学的就是这个专业啊。但是专业名称和学生其实没多大关系，专业名称依赖于专业编号。这就叫传递依赖，就是某一个字段不直接依赖主键，而是依赖 依赖主键的另一个字段。

解决方法：

学生表，学生编号为主键：

学生编号为主键：

学生编号	学生姓名	专业编号
1001	白杰	2001
1002	杨春旺	2002
1003	张志伟	2001

专业表，专业编号为主键：

专业编号	专业名称
2001	计算机
2002	自动化

以上设计是典型的一对多的设计，一存储在一张表中，多存储在一张表中，**在多的那张表中添加外键指向一的一方。**

二、常见表关系

一对一 用的不多

一个表和另一张表存在的关系是一对一，此种设计不常用，应为此种关系经常会将多张表合并为一张表。

举例：

学生信息表可以分为基本信息表，和详细信息表。

可能有这种需求，需要给个某个账户对学生表的操作，但是有些私密信息又不能暴露，就可以拆分。

```
第一种方案：分两张表存储，共享主键
第二种方案：分两张表存储，外键唯一
```

一对多

第三范式的例子

两张表　外键建在多的一方

> 分两张表存储，在多的一方添加外键，
> 这个外键字段引用一的一方中的主键字段

多对多

第二范式的例子

> 分三张表存储，在学生表中存储学生信息，在课程表中存储课程信息，
> 在成绩表中存储学生和课程的关系信息