Table of Contents

为什么学习数据库？

1、90%以上的软件都需要操作数据，比如游戏、社交、新闻、商城、财务等，这些软件都在不停的展示、存储数据，它们的数据都存储在数据库，数据库是软件的基础。

2、测试工程师在测试软件的过程中，不仅需要在界面上操作，还需要检查数据库中的数据是否正确。从而在软件出问题时，测出更深层的问题。

例如：测试注册登录功能，在输入了注册信息后，提示注册成功，但是使用刚才注册的信息登录不成功。 这时需要检查数据库中是否保存了正确的注册信息，如果数据库中没有保存数据，那么使用刚才注册的账号肯定登录不了，这样就能定位问题：注册时，没有把数据存储起来。

数据库阶段知识点

1. 数据库的基本概念
2. Navicat 操作数据库
3. SQL 语言（重点）
4. MySQL 高级（了解）
5. Redis

学习目标

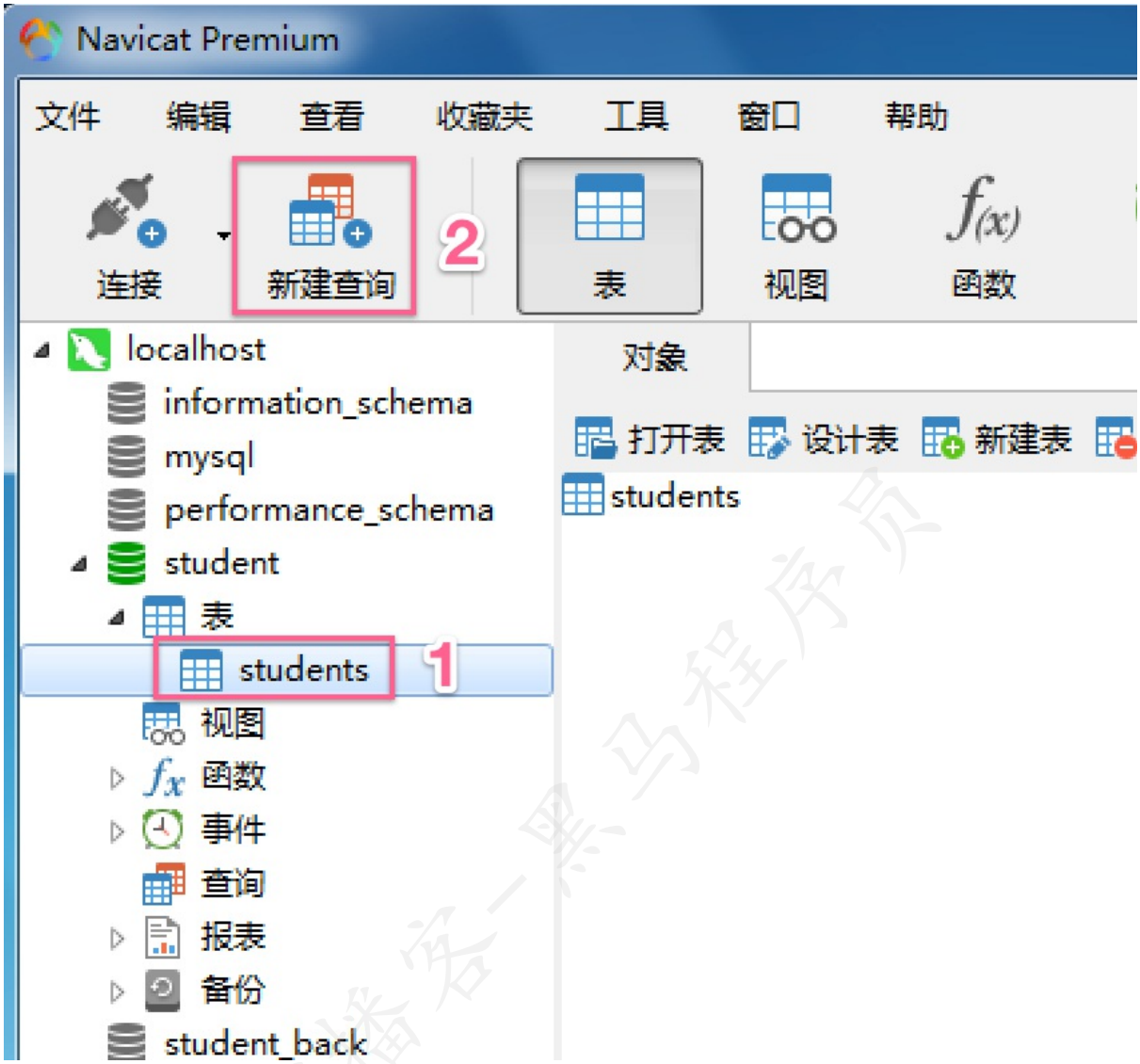
- 熟练编写 SQL 语言中的查询语句

SQL 语言

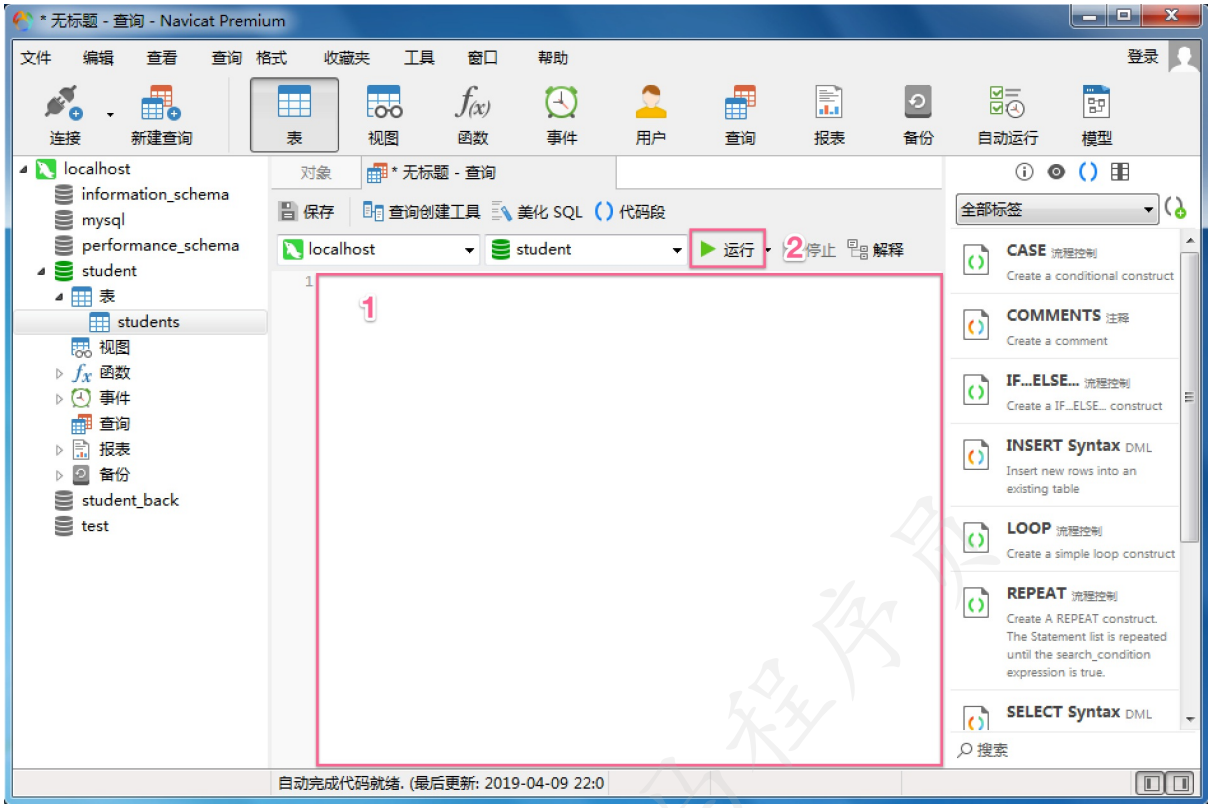
主要操作

- 数据表操作
  - 创建表
  - 删除表
- 数据操作
  - 增加数据
  - 删除数据
  - 修改数据
  - 查询数据（重点）

SQL 语言的编写和运行



1. 选择表名
2. 选择 新建查询



1. 编写 SQL 语句
2. 运行 查看结果

数据表操作

1. 创建表

```
create table 表名(
    字段名 类型 约束,
    字段名 类型 约束
    ...
)
```

例：创建学生表，字段要求如下：

姓名(长度为10)

```
create table students(
    name varchar(10)
)
```

例：创建学生表，字段要求如下：

姓名(长度为10)， 年龄

```
create table students(
    name varchar(10),
    age int unsigned
)
```

例：创建学生表，字段要求如下：

姓名(长度为10)， 年龄，身高(保留小数点2位)

```
create table students(
    id int unsigned primary key auto_increment,
    name varchar(20),
    age int unsigned,
    height decimal(5,2)
)
```

2. 删除表

```
格式一：drop table 表名
```

格式二：drop table if exists 表名

例：删除学生表

```
drop table students
或
drop table if exists students
```

数据操作-增删改查

1. 简单查询

```
select * from 表名
例：查询所有学生数据
select * from students
```

2. 添加数据

2.1 添加一行数据

格式一：所有字段设置值，值的顺序与表中字段的顺序对应

- 说明：主键列是自动增长，插入时需要占位，通常使用0或者 default 或者 null 来占位，插入成功后以实际数据为准

```
insert into 表名 values(...)
```

例：插入一个学生，设置所有字段的信息

```
insert into students values(0,'亚瑟',22,177.56)
```

格式二：部分字段设置值，值的顺序与给出的字段顺序对应

```
insert into 表名(字段1,...) values(值1,...)
```

例：插入一个学生，只设置姓名

```
insert into students(name) values('老夫子')
```

2.2 添加多行数据

方式一：写多条**insert**语句，语句之间用英文分号隔开

```
insert into students(name) values('老夫子2');
insert into students(name) values('老夫子3');
insert into students values(0,'亚瑟2',23,167.56)
```

方式二：写一条**insert**语句，设置多条数据，数据之间用英文逗号隔开

```
格式一：insert into 表名 values(...),(...)...
例：插入多个学生，设置所有字段的信息
insert into students values(0,'亚瑟3',23,167.56),(0,'亚瑟4',23,167.56)
```

```
格式二：insert into 表名(列1,...) values(值1,...),(值1,...)...
例：插入多个学生，只设置姓名
insert into students(name) values('老夫子5'),('老夫子6')
```

3. 修改

```
update 表名 set 列1=值1,列2=值2... where 条件
```

例：修改id为5的学生数据，姓名改为 狄仁杰，年龄改为 20

```
update students set name='狄仁杰',age=20 where id=5
```

4. 删除

```
格式一：delete from 表名 where 条件
```

例：删除id为6的学生数据

```
delete from students where id=6
```

逻辑删除：对于重要的数据，不能轻易执行 **delete** 语句进行删除。因为一旦删除，数据无法恢复，这时可以进行逻辑删除。

1、给表添加字段，代表数据是否删除，一般起名 isdelete，0代表未删除，1代表删除，默认值为0

2、当要删除某条数据时，只需要设置这条数据的 isdelete 字段为1

3、以后在查询数据时，只查询出 isdelete 为0的数据

```
例：
1、给学生表添加字段(isdelete)，默认值为0，
如果表中已经有数据，需要把所有数据的isdelete字段更新为0
update students set isdelete=0
2、删除id为1的学生
update students set isdelete=1 where id=1
3、查询未删除的数据
```

```
select * from students where isdelete=0
```

格式二：truncate table 表名（删除表的所有数据，保留表结构）

例：删除学生表的所有数据

```
truncate table students
```

格式三：drop table 表名（删除表，所有数据和表结构都删掉）

例：删除学生表

```
drop table students
```

Truncate、Delete、Drop 的区别

1、Delete 删除数据时，即使删除所有数据，其中的自增长字段不会从1开始

2、Truncate 删除数据时，其中的自增长字段恢复从1开始

3、Drop 是删除表，所有数据和表结构都删掉

总结

- 在速度上，drop > truncate > delete
- 如果想删除部分数据用 delete，注意带上 where 子句
- 如果想删除表，用 drop
- 如果想保留表而将所有数据删除，自增长字段恢复从1开始，用 truncate

数据操作-查询

1. 数据准备

1.1 创建数据表

```
drop table if exists students;
create table students (
  studentNo varchar(10) primary key,
  name varchar(10),
  sex varchar(1),
  hometown varchar(20),
  age tinyint(4),
  class varchar(10),
  card varchar(20)
);
```

1.2 插入数据

```
insert into students values
('001', '王昭君', '女', '北京', '20', '1班', '340322199001247654'),
('002', '诸葛亮', '男', '上海', '18', '2班', '340322199002242354'),
('003', '张飞', '男', '南京', '24', '3班', '340322199003247654'),
('004', '白起', '男', '安徽', '22', '4班', '340322199005247654'),
('005', '大乔', '女', '天津', '19', '3班', '340322199004247654'),
('006', '孙尚香', '女', '河北', '18', '1班', '340322199006247654'),
('007', '百里玄策', '男', '山西', '20', '2班', '340322199007247654'),
('008', '小乔', '女', '河南', '15', '3班', null),
('009', '百里守约', '男', '湖南', '21', '1班', ''),
('010', '妲己', '女', '广东', '26', '2班', '340322199607247654'),
('011', '李白', '男', '北京', '30', '4班', '340322199005267754'),
('012', '孙膑', '男', '新疆', '26', '3班', '340322199000297655');
```

---

2. 查询基本语法

2.1 查询所有字段

```
语法：
select * from 表名
```

例：查询所有学生的所有字段
```
select * from students
```

2.2 查询部分字段

语法：
```
select 字段1,字段2,... from 表名
```

例：查询所有学生的姓名、性别、年龄
```
select name,sex,age from students
```

2.3 起别名

给表起别名，在多表查询中经常使用
语法：
```
select 别名.字段1,别名.字段2,... from 表名 as 别名
```

例：给学生表起别名
```
select s.name,s.sex,s.age from students as s;
```

给字段起别名，这个别名出现在结果集中
语法：
```
select 字段1 as 别名1,字段2 as 别名2,... from 表名
```

例：查询所有学生的姓名、性别、年龄，结果中的字段名显示为中文
```
select name as 姓名,sex as 性别,age as 年龄 from students;
```

2.4 去重

语法：
```
select distinct 字段1,... from 表名
```

例：查询所有学生的性别，不显示重复的数据
```
select distinct sex from students;
```

条件查询

1. 语法格式

使用 where 子句对表中的数据筛选，符合条件的数据会出现在结果集中

1.1 语法

```
select 字段1,字段2... from 表名 where 条件;

-- eg:
select * from students where id=1;
```

where 后面支持多种运算符，进行条件的处理

- 比较运算
- 逻辑运算
- 模糊查询
- 范围查询
- 空判断

1.2 比较运算符

- 等于: =
- 大于: >
- 大于等于: >=
- 小于: <
- 小于等于: <=
- 不等于: != 或 <>

例1：查询小乔的年龄

```
select age from students where name='小乔'
```

例2：查询20岁以下的学生

```
select * from students where age<20
```

例3：查询家乡不在北京的学生

```
select * from students where hometown!='北京'
```

练习：

```
1、查询学号是'007'的学生的身份证号
2、查询'1班'以外的学生信息
3、查询年龄大于20的学生的姓名和性别
```

1.3 逻辑运算符

- and
- or
- not

例1：查询年龄小于20的女同学

```
select * from students where age<20 and sex='女'
```

例2：查询女学生或'1班'的学生

```
select * from students where sex='女' or class='1班'
```

例3：查询非天津的学生

```
select * from students where not hometown='天津'
```

练习：

```
1、查询河南或河北的学生
2、查询'1班'的'上海'的学生
3、查询非20岁的学生
```

1.4 模糊查询

- like
- %表示任意多个任意字符
- _表示一个任意字符

例1：查询姓孙的学生

```
select * from students where name like '孙%'
```

例2：查询姓孙且名字是一个字的学生

```
select * from students where name like '孙_'
```

例3：查询姓名以乔结尾的学生

```
select * from students where name like '%乔'
```

例4：查询姓名含白的学生

```
select * from students where name like '%白%'
```

练习：

```
1、查询姓名为两个字的学生
2、查询姓百且年龄大于20的学生
3、查询学号以1结尾的学生
```

1.5 范围查询

- in表示在一个非连续的范围内

例1：查询家乡是北京或上海或广东的学生

```
select * from students where hometown in('北京','上海','广东')
```

- between ... and ...表示在一个连续的范围内

例2：查询年龄为18至20的学生

```
select * from students where age between 18 and 20
```

练习：

```
1、查询年龄在18或19或22的女生
2、查询年龄在20到25以外的学生
```

1.6 空判断

- 注意：null与''是不同的
- 判空is null

例1：查询没有填写身份证的学生

```
select * from students where card is null
```

- 判非空is not null

例2：查询填写了身份证的学生

```
select * from students where card is not null
```

排序

1. 排序

为了方便查看数据，可以对数据进行排序

1.1 语法

```
select * from 表名
order by 列1 asc|desc,列2 asc|desc,...
```

- 将行数据按照列1进行排序，如果某些行列1的值相同时，则按照列2排序，以此类推
- 默认按照列值从小到大排列
- asc从小到大排列，即升序
- desc从大到小排序，即降序

1.2 示例

例1：查询所有学生信息，按年龄从小到大排序

```
select * from students order by age
```

例2：查询所有学生信息，按年龄从大到小排序，年龄相同时，再按学号从小到大排序

```
select * from students order by age desc,studentNo
```

练习：

```
1、查询所有学生信息，按班级从小到大排序，班级相同时，再按学号再按学号从小到大排序
```

聚合函数

1. 聚合函数

- 使用聚合函数方便进行数据统计
- 聚合函数不能在 where 中使用

常用聚合函数

- count(): 查询总记录数
- max(): 查询最大值
- min(): 查询最小值
- sum(): 求和
- avg(): 求平均值

1.1 查询总记录数

count(*)表示计算总行数，括号中也可以使用字段名

示例：查询学生总数

```
select count(*) from students;
```

1.2 查询最大值

max(列)表示求此列的最大值

示例：查询女生的最大年龄

```
select max(age) from students where sex='女';
```

1.3 查询最小值

min(列)表示求此列的最小值

示例：查询1班的最小年龄

```
select min(age) from students;
```

1.4 求和

sum(列)表示求此列的和

示例：查询北京学生的年龄总和

```
select sum(age) from students where hometown='北京';
```

1.5 求平均值

avg(列)表示求此列的平均值

示例：查询女生的平均年龄

```
select avg(age) from students where sex='女'
```

1.6 练习

```
1、查询所有学生的最大年龄、最小年龄、平均年龄
2、一班共有多少个学生
3、查询3班年龄小于18岁的同学有几个
```

分组

1. 分组

- 按照字段分组，此字段相同的数据会被放到一个组中
- 分组的目的是对每一组的数据进行统计(使用聚合函数)

1.1 语法

```
select 字段1,字段2,聚合... from 表名 group by 字段1,字段2...
```

例1：查询各种性别的人数

```
select sex,count(*) from students group by sex
```

例2：查询每个班级中各种性别的人数

```
select class,sex,count(*) from students group by class,sex
```

练习

```
1、查询各个班级学生的平均年龄、最大年龄、最小年龄
```

2. 分组后的数据筛选

2.1 语法

```
select 字段1,字段2,聚合... from 表名
group by 字段1,字段2,字段3...
having 字段1,...聚合...
```

- having后面的条件运算符与where的相同

例1：查询男生总人数

```
方案一
select count(*) from students where sex='男'
-----------------------------------
方案二：
```

```
select sex,count(*) from students group by sex having sex='男'
```

练习

```
1、查询1班除外其他班级学生的平均年龄、最大年龄、最小年龄
```

2.2 对比 where 与 having

- where 是对 from 后面指定的表进行数据筛选，属于对原始数据的筛选
- having 是对 group by 的结果进行筛选
- having 后面的条件中可以用聚合函数，where 后面不可以

分页

1. 获取部分行

当数据量过大时，在一页中查看数据是一件非常麻烦的事情

1.1 语法

```
select * from 表名 limit start,count
```

- 从start开始，获取count条数据
- start索引从0开始

例1：查询前3行学生信息

```
select * from students limit 0,3
```

练习：

```
1、查询第4到第6行学生信息
```

2. 分页

limit典型的应用场景就是实现分页查询

已知：每页显示m条数据，求：显示第n页的数据

```
select * from students limit (n-1)*m,m
```

练习：

```
1、每页显示5条数据，显示每一页的数据
```

> 扩展：已知总记录数和每页显示条数，求总页数？

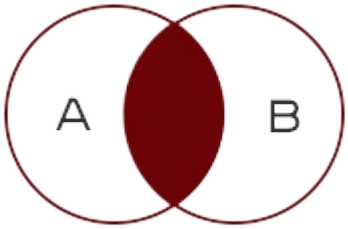
连接查询

1. 连接查询

当查询结果的列来源于多张表时，需要将多张表连接成一个大的数据集，再选择合适的列返回

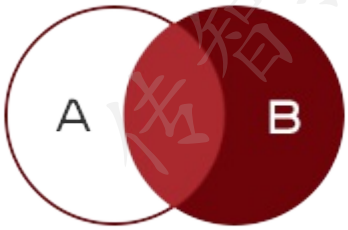
1.1 常用的连接方式

- 内连接：查询的结果为两个表匹配到的数据



- 左连接：查询的结果为两个表匹配到的数据加左表特有的数据，对于右表中不存在的数据使用 null 填充

- 右连接：查询的结果为两个表匹配到的数据加右表特有的数据，对于左表中不存在的数据使用 null 填充



2. 准备数据

```
drop table if exists courses;
create table courses (
  courseNo int(10) unsigned primary key auto_increment,
  name varchar(10)
);

insert into courses values
('1', '数据库'),
```

```
('2', 'qtp'),
('3', 'linux'),
('4', '系统测试'),
('5', '单元测试'),
('6', '测试过程');

drop table if exists scores;
create table scores (
  id int(10) unsigned primary key auto_increment,
  courseNo int(10),
  studentno varchar(10),
  score tinyint(4)
);

insert into scores values
('1', '1', '001', '90'),
('2', '1', '002', '75'),
('3', '2', '002', '98'),
('4', '3', '001', '86'),
('5', '3', '003', '80'),
('6', '4', '004', '79'),
('7', '5', '005', '96'),
('8', '6', '006', '80');
```

3. 内连接

语法：

```
select * from 表1
inner join 表2 on 表1.列=表2.列
```

例1：查询学生信息及学生的成绩

```
select
    *
from
    students stu
inner join scores sc on stu.studentNo = sc.studentNo
```

扩展：内连接的另一种语法

```
select * from 表1,表2 where 表1.列=表2.列

例1：查询学生信息及学生的成绩
```

```
select
    *
from
    students stu,
    scores sc
where
    stu.studentNo = sc.studentNo
```

例2：查询课程信息及课程的成绩

```
select
    *
from
    courses cs
inner join scores sc on cs.courseNo = sc.courseNo
```

例3：查询学生信息及学生的课程对应的成绩

```
select
    *
from
    students stu
inner join scores sc on stu.studentNo = sc.studentNo
inner join courses cs on cs.courseNo = sc.courseNo
```

例4：查询王昭君的成绩，要求显示姓名、课程号、成绩

```
select
    stu.name,
    sc.courseNo,
    sc.score
from
    students stu
inner join scores sc on stu.studentNo = sc.studentNo
where
    stu.name = '王昭君'
```

例5：查询王昭君的数据库成绩，要求显示姓名、课程名、成绩

```
select
    stu.name,
    cs.name,
    sc.score
from
    students stu
```

```
inner join scores sc on stu.studentNo = sc.studentNo
inner join courses cs on sc.courseNo = cs.courseNo
where
    stu.name = '王昭君' and cs.name = '数据库'
```

例6：查询所有学生的数据库成绩，要求显示姓名、课程名、成绩

```
select
    stu.name,
    cs.name,
    sc.score
from
    students stu
inner join scores sc on stu.studentNo = sc.studentNo
inner join courses cs on sc.courseNo = cs.courseNo
where
    cs.name = '数据库'
```

例7：查询男生中最高成绩，要求显示姓名、课程名、成绩

```
select
    stu.name,
    cs.name,
    sc.score
from
    students stu
inner join scores sc on stu.studentNo = sc.studentNo
inner join courses cs on sc.courseNo = cs.courseNo
where
    stu.sex = '男'
order by
    sc.score desc
limit 1
```

4. 左连接

语法：

```
select * from 表1
left join 表2 on 表1.列=表2.列
```

例1：查询所有学生的成绩，包括没有成绩的学生

```
select
    *
```

```
from
    students stu
left join scores sc on stu.studentNo = sc.studentNo
```

例2：查询所有学生的成绩，包括没有成绩的学生，需要显示课程名

```
select
    *
from
    students stu
left join scores sc on stu.studentNo = sc.studentNo
left join courses cs on cs.courseNo = sc.courseNo
```

5. 右连接

语法：

```
select * from 表1
right join 表2 on 表1.列=表2.列
```

例1：查询所有学生的成绩，包括没有成绩的学生

```
select
    *
from
    scores sc
right join students stu on stu.studentNo = sc.studentNo
```

例2：查询所有学生的成绩，包括没有成绩的学生，需要显示课程名

```
select
    *
from
    scores sc
right join students stu on stu.studentNo = sc.studentNo
left join courses cs on cs.courseNo = sc.courseNo
```

自关联

1. 案例

- 设计省信息的表结构provinces
  - id
  - ptitle
- 设计市信息的表结构citys
  - id
  - ctitle
  - proid
- citys表的proid表示城市所属的省，对应着provinces表的id值
- 问题：能不能将两个表合成一张表呢？
- 思考：观察两张表发现，citys表比provinces表多一个列proid，其它列的类型都是一样的
- 意义：存储的都是地区信息，而且每种信息的数据量有限，没必要增加一个新表，或者将来还要存储区、乡镇信息，都增加新表的开销太大
- 答案：定义表areas，结构如下
  - id
  - atitle
  - pid
- 因为省没有所属的省份，所以可以填写为null
- 城市所属的省份pid，填写省所对应的编号id
- 这就是自关联，表中的某一列，关联了这个表中的另外一列，但是它们的业务逻辑含义是不一样的，城市信息的pid引用的是省信息的id
- 在这个表中，结构不变，可以添加区县、乡镇街道、村社区等信息

准备数据：

```
drop table if exists areas;
create table areas(
    aid int primary key,
    atitle varchar(20),
    pid int
);

insert into areas values
('130000', '河北省', NULL),
('130100', '石家庄市', '130000'),
('130400', '邯郸市', '130000'),
('130600', '保定市', '130000'),
('130700', '张家口市', '130000'),
('130800', '承德市', '130000'),
('410000', '河南省', NULL),
```

```
('410100', '郑州市', '410000'),
('410300', '洛阳市', '410000'),
('410500', '安阳市', '410000'),
('410700', '新乡市', '410000'),
('410800', '焦作市', '410000'),
('410101', '中原区', '410100'),
('410102', '二七区', '410100'),
('410301', '洛龙区', '410300');
```

2. 自关联

自关联：表中的某一列，关联了这个表中的另外一列，但是它们的业务逻辑含义是不一样的。（自己关联自己）

例1：查询河南省所有的市

```
select
    *
from
    areas as a1
inner join areas as a2 on a1.aid=a2.pid
where
    a1.atitle='河南省';
```

例2：查询郑州市的所有的区

```
select
    *
from
    areas as a1
inner join areas as a2 on a1.aid=a2.pid
where
    a1.atitle='郑州市';
```

例3：查询河南省的所有的市区

```
select
    *
from
    areas as a1
inner join areas as a2 on a1.aid=a2.pid
inner join areas as a3 on a2.aid=a3.pid
where
    a1.atitle='河南省'
```

子查询

1. 子查询介绍

子查询：在一个 select 语句中，嵌入了另外一个 select 语句，那么嵌入的 select 语句称之为子查询语句

主查询：外层的 select 语句称之为主查询语

1.1 主查询和子查询的关系

- 子查询是嵌入到主查询中的
- 子查询是辅助主查询的，要么充当条件，要么充当数据源
- 子查询是可以独立使用的语句，是一条完整的 select 语句

2. 子查找充当条件

例1：查询大于平均年龄的学生

```
查询班级学生平均年龄
select avg(age) from students

查询大于平均年龄的学生
select * from students where age > 21.4167

select * from students where age > (select avg(age) from students);
```

例2：查询王昭君的成绩，要求显示成绩

```
学生表中查询王昭君的学号
select studentNo from students where name = '王昭君'

成绩表中根据学号查询成
select * from scores where studentNo = '001'

select * from scores
where studentNo = (select studentNo from students where name = '王昭君')
```

例**1**和例**2**中：子查询返回的结果只有一个值**(**一行一列**)**，这种称之为标量子查询

例3：查询18岁的学生的成绩，要求显示成绩

```
学生表中查询18岁的学生的学号
```

```
select studentNo from students where age=18

成绩表中根据学号查询成绩
select * from scores where studentNo in ('002','006')

select * from scores
where studentNo in (select studentNo from students where age=18)
```

例**3**中：子查询返回的结果是一列数据**(**一列多行**)**，这种称之为列子查询

例4：查询和王昭君同班、同龄的学生信息

```
select class,age from students where name='王昭君'

select * from students where class='1班' and age=20

select * from students where (class,age)=('1班',20)

select * from students
where (class,age)=(select class,age from students where name='王昭君')
```

例**4**中：子查询返回的结果是一行**(**一行多列**)**，这种称之为行子查询

3. 子查询充当数据源

例5：查询数据库和系统测试的课程成绩

```
select
    *
from
    scores s
inner join
    (select * from courses where name in ('数据库','系统测试')) c
on s.courseNo = c.courseNo
```

例**5**中：子查询返回的结果是多行多列**(**相当于一个表**)**，这种称之为表级子查询

4. 子查询中特定关键字使用

- in 范围
  - 格式: 主查询 where 条件 in (列子查询)
- any | some 任意一个
  - 格式: 主查询 where 列 = any (列子查询)

    - 在条件查询的结果中匹配任意一个即可,等价于 in
- all
    - 格式: 主查询 where 列 = all(列子查询) : 等于里面所有
    - 格式: 主查询 where 列 <>all(列子查询) : 不等一其中所有

```
select * from students
where age in (select age from students where age between 18 and 20)
```

查询演练

1. 数据准备

创建表

- 部门 departments
- 员工 employees
- 工资表 salary

部门表

```
drop table if exists departments;
create table departments (
  deptid int(10) primary key,
  deptname varchar(20) not null -- 部门名称
);

insert into departments values ('1001', '市场部');
insert into departments values ('1002', '测试部');
insert into departments values ('1003', '开发部');
```

员工表

```
drop table if exists employees;
create table employees (
  empid int(10) primary key,
  empname varchar(20) not null, -- 姓名
  sex varchar(4) default null, -- 性别
  deptid int(20) default null, -- 部门编号
  jobs varchar(20) default null, -- 岗位
  politicalstatus varchar(20) default null, -- 政治面貌
  leader int(10) default null
);

insert into employees values ('1', '王昭君', '女', '1003', '开发', '群众', '9');
insert into employees values ('2', '诸葛亮', '男', '1003', '开发经理', '群众', null);
insert into employees values ('3', '张飞', '男', '1002', '测试', '团员', '4');
insert into employees values ('4', '白起', '男', '1002', '测试经理', '党员', null);
insert into employees values ('5', '大乔', '女', '1002', '测试', '党员', '4');
insert into employees values ('6', '孙尚香', '女', '1001', '市场', '党员', '12');
insert into employees values ('7', '百里玄策', '男', '1001', '市场', '团员', '12');
insert into employees values ('8', '小乔', '女', '1002', '测试', '群众', '4');
insert into employees values ('9', '百里守约', '男', '1003', '开发', '党员', '9');
```

```
insert into employees values ('10', '妲己', '女', '1003', '开发', '团员', '9');
insert into employees values ('11', '李白', '男', '1002', '测试', '团员', '4');
insert into employees values ('12', '孙膑', '男', '1001', '市场经理', '党员', null);
```

工资表

```
drop table if exists salary;
create table salary (
  sid int(10) primary key,
  empid int(10) not null,
  salary int(10) not null -- 工资
);

insert into salary values ('1', '7', '2100');
insert into salary values ('2', '6', '2000');
insert into salary values ('3', '12', '5000');
insert into salary values ('4', '9', '1999');
insert into salary values ('5', '10', '1900');
insert into salary values ('6', '1', '3000');
insert into salary values ('7', '2', '5500');
insert into salary values ('8', '5', '2000');
insert into salary values ('9', '3', '1500');
insert into salary values ('10', '8', '4000');
insert into salary values ('11', '11', '2600');
insert into salary values ('12', '4', '5300');
```

2. 练习

1）列出总人数大于**4**的部门号和总人数

```
select deptid, count(*) from employees e group by deptid having count(*)>4
```

2）列出开发部和和测试部的职工号、姓名

```
select e.empid,d.empname from employees e
inner join departments d on e.deptid = d.deptid
where d.deptname in ('开发部','测试部')
```

3）求出各部门党员的人数，要求显示部门名称

```
select d.deptname,count(*) from employees e
inner join departments d on e.deptid=d.deptid
where e.politicalstatus = '党员' group by e.deptid
```

4）列出市场部的所有女职工的姓名和政治面貌

```
select e.empname,e.politicalstatus from employees e
inner join departments d on e.deptid = d.deptid
where e.sex= '女' and d.deptname = '市场部'
```

5）显示所有职工的姓名、部门名和工资数

```
select e.empname,d.deptname,s.salary from employees e
inner join departments d on e.deptid = d.deptid
inner join salary s on e.empid = s.empid
```

6）显示各部门名和该部门的职工平均工资

```
select d.deptname, avg(s.salary)  from  departments d
inner join employees e on d.deptid = e.deptid
inner join salary s on e.empid = s.empid group by d.deptname
```

7）显示工资最高的前**3**名职工的职工号和姓名

```
select e.empid, e.empname,s.salary from salary s
inner join employees e on s.empid = e.empid order by s.salary desc limit 3
```

8）列出工资在**1000**－**2000**之间的所有职工姓名

```
select e.empname,s.salary from salary s
inner join employees e on s.empid = e.empid
where s.salary between 1000 and 2000
```

9）列出工资比王昭君高的员工

```
select * from employees e
inner join salary s on e.empid=s.empid
where s.salary>
(select s.salary from employees e
inner join salary s on e.empid=s.empid
where e.empname='王昭君')
```

10）列出每个部门中工资小于本部门平均工资的员工信息

```
select * from employees e
inner join salary s on e.empid=s.empid
inner join
```

```
(select d.deptid,d.deptname, avg(s.salary) avg_salary  from  departments d
inner join employees e on d.deptid = e.deptid
inner join salary s on e.empid = s.empid group by d.deptname) temp
on e.deptid=temp.deptid
where s.salary<temp.avg_salary
```