15.1 查询

创建数据库、数据表

```
-- 创建数据库
create database python_test_1 charset=utf8;

-- 使用数据库
use python_test_1;

-- students 表
create table students(
    id int unsigned primary key auto_increment not null,
    name varchar(20) default '',
    age tinyint unsigned default 0,
    height decimal(5,2),
    gender enum('男','女','中性','保密') default '保密',
    cls_id int unsigned default 0,
    is_delete bit default 0
);

-- classes 表
create table classes (
    id int unsigned auto_increment primary key not null,
    name varchar(30) not null
);
```

准备数据

```
-- 向 students 表中插入数据
INSERT INTO students(name,age,height,gender,cls_id,is_delete)

VALUES

      ( '小明', 18, 180.00, 2, 1, 0 ),

      ( '小月月', 18, 180.00, 2, 2, 1 ),

      ( '彭于晏', 29, 185.00, 1, 1, 0 ),

      ( '刘德华', 59, 175.00, 1, 2, 1 ),

      ( '黄蓉', 38, 160.00, 2, 1, 0 ),

      ( '凤姐', 28, 150.00, 4, 2, 1 ),

      ( '王祖贤', 18, 172.00, 2, 1, 1 ),

      ( '周杰伦', 36, NULL, 1, 1, 0 ),
```

```
    ( '程坤', 27, 181.00, 1, 2, 0 ),
    ( '刘亦菲', 25, 166.00, 2, 2, 0 ),
    ( '金星', 33, 162.00, 3, 3, 1 ),
    ( '静香', 12, 180.00, 2, 4, 0 ),
    ( '郭靖', 12, 170.00, 1, 4, 0 ),
    ( '周杰', 34, 176.00, 2, 5, 0 );

-- 向 classes 表中插入数据
insert into classes values (0, "python_01 期"), (0, "python_02 期");
```

- 查询所有字段

```
select * from 表名;
例:
select * from students;
```

- 查询指定字段

```
select 列 1,列 2,... from 表名;
例:
select name from students;
```

- **使用 as 给字段起别名**

```
select id as 序号, name as 名字, gender as 性别 from students;
```

- 可以通过 as 给表起别名

```
-- 如果是单表查询 可以省略表明
select id, name, gender from students;

-- 表名.字段名
select students.id,students.name,students.gender from students;

-- 可以通过 as 给表起别名
select s.id,s.name,s.gender from students as s;
```

消除重复行

- 在 select 后面列前使用 distinct 可以消除重复的行

```
select distinct 列 1,... from 表名;
例:
select distinct gender from students;
```

15.2 条件

使用 where 子句对表中的数据筛选，结果为 true 的行会出现在结果集中

- 语法如下：

```
select * from 表名 where 条件;
```

例：

```
select * from students where id=1;
```

- where 后面支持多种运算符，进行条件的处理
  - **比较运算符**
  - 逻辑运算符
  - 模糊查询
  - 范围查询
  - 空判断

比较运算符

- 是否等于: =
- 大于: >
- 大于等于: >=
- 小于: <
- 小于等于: <=
- 不等于: != 或 <> **低效率操作，不要使用**

例 1：查询编号大于 3 的学生

```
select * from students where id > 3;
```

例 2：查询编号不大于 4 的学生

```
select * from students where id <= 4;
```

例 3：查询姓名不是“黄蓉”的学生

```
select * from students where name != '黄蓉';
```

例 4：查询没被删除的学生

```
select * from students where is_delete=0;
```

逻辑运算符

- and
- or

- not　这个不好，不要用

例 5：查询编号大于 3 的女同学

```
select * from students where id > 3 and gender=2;
```

例 6：查询编号小于 4 或没被删除的学生

```
select * from students where id < 4 or is_delete=0;
```

模糊查询

- like
- %表示零个或任意多个任意字符
- _表示一个　任意字符

例 7：查询姓黄的学生

```
select * from students where name like '黄%';
```

例 8：查询姓黄并且“名”是一个字的学生

```
select * from students where name like '黄_';
```

例 9：查询姓黄或叫靖的学生

```
select * from students where name like '黄%' or name like '%靖';
```

范围查询

- in 表示在一个非连续的范围内

例 10：查询编号是 1 或 3 或 8 的学生

```
select * from students where id in(1,3,8);
```

- between ... and ...表示在一个连续的范围内，闭区间

例 11：查询编号为 3 至 8 的学生

```
select * from students where id between 3 and 8;
```

例 12：查询编号是 3 至 8 的男生

```
select * from students where id between 3 and 8 and gender=1;
```

空判断

- 注意：null 与''是不同的
- **判空 is null**

例 13：查询没有填写身高的学生

```
select * from students where height is null;
```

- 判非空 is not null

例 14：查询填写了身高的学生

```
select * from students where height is not null;
```

例 15：查询填写了身高的男生

```
select * from students where height is not null and gender=1;
```

优先级

- 优先级由高到低的顺序为：**小括号，not，比较运算符，逻辑运算符**
- **and 比 or 先运算，如果同时出现并希望先算 or，需要结合()使用**

15.3 排序

为了方便查看数据，可以对数据进行排序

语法：

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

说明

- 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）**
- asc 从小到大排列，即升序
- desc 从大到小排序，即降序

例 1：查询未删除男生信息，按学号降序

```
select * from students where gender=1 and is_delete=0 order by id desc;
```

例 2：查询未删除学生信息，按名称升序

```
select * from students where is_delete=0 order by name;
```

例 3：显示所有的学生信息，先按照年龄从大-->小排序，当年龄相同时 按照身高从高-->矮排序

```
select * from students  order by age desc,height desc;
```

15.4 聚合函数

为了快速得到统计数据，经常会用到如下 5 个聚合函数

总数

- count(*)表示计算总行数，括号中写星与列名，结果是相同的

例 1：查询学生总数

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

最大值

- max(列)表示求此列的最大值

例 2：查询女生的编号最大值

```
select max(id) from students where gender=2;
```

最小值

- min(列)表示求此列的最小值

例 3：查询未删除的学生最小编号

```
select min(id) from students where is_delete=0;
```

求和

- sum(列)表示求此列的和

例 4：查询男生的总年龄

```
select sum(age) from students where gender=1;

-- 平均年龄
select sum(age)/count(*) from students where gender=1;
```

平均值

- avg(列)表示求此列的平均值

例 5：查询未删除女生的编号平均值

```
select avg(id) from students where is_delete=0 and gender=2;
```

15.5 分组

group by

1. group by 的含义:将查询结果按照 1 个或多个字段进行分组，字段值相同的为一组
2. group by 可用于单个字段分组，也可用于多个字段分组

```
select * from students;
+----+-----------+------+--------+--------+--------+-----------+
| id | name      | age  | height | gender | cls_id | is_delete |
+----+-----------+------+--------+--------+--------+-----------+
|  1 | 小明      |   18 | 180.00 | 女     |      1 |           |
|  2 | 小月月    |   18 | 180.00 | 女     |      2 |           |
|  3 | 彭于晏    |   29 | 185.00 | 男     |      1 |           |
|  4 | 刘德华    |   59 | 175.00 | 男     |      2 |           |
|  5 | 黄蓉      |   38 | 160.00 | 女     |      1 |           |
|  6 | 凤姐      |   28 | 150.00 | 保密   |      2 |           |
|  7 | 王祖贤    |   18 | 172.00 | 女     |      1 |           |
|  8 | 周杰伦    |   36 |   NULL | 男     |      1 |           |
|  9 | 程坤      |   27 | 181.00 | 男     |      2 |           |
| 10 | 刘亦菲    |   25 | 166.00 | 女     |      2 |           |
| 11 | 金星      |   33 | 162.00 | 中性   |      3 |           |
| 12 | 静香      |   12 | 180.00 | 女     |      4 |           |
| 13 | 周杰      |   34 | 176.00 | 女     |      5 |           |
| 14 | 郭靖      |   12 | 170.00 | 男     |      4 |           |
+----+-----------+------+--------+--------+--------+-----------+

select gender from students group by gender;
+--------+
| gender |
+--------+
| 男     |
| 女     |
| 中性   |
| 保密   |
+--------+
```

根据 gender 字段来分组，gender 字段的全部值有 4 个'男','女','中性','保密'，所以分为了 4 组 当 group by 单独使用时，只显示出每组的第一条记录, 所以 group by 单独使用时的实际意义不大

group by + group_concat()

1. group_concat(字段名)可以作为一个输出字段来使用，concat 是构造合并的含义

2. 表示分组之后，根据分组结果，使用 group_concat()来放置每一组的某字段的值的集合

```
select gender from students group by gender;
+--------+
| gender |
+--------+
| 男     |
| 女     |
| 中性   |
| 保密   |
+--------+

select gender,group_concat(name) from students group by gender;
+--------+-----------------------------------------------------------+
| gender | group_concat(name)                                        |
+--------+-----------------------------------------------------------+
| 男     | 彭于晏,刘德华,周杰伦,程坤,郭靖                                  |
| 女     | 小明,小月月,黄蓉,王祖贤,刘亦菲,静香,周杰                          |
| 中性   | 金星                                                      |
| 保密   | 凤姐                                                      |
+--------+-----------------------------------------------------------+


select gender,group_concat(id) from students group by gender;
+--------+------------------+
| gender | group_concat(id) |
+--------+------------------+
| 男     | 3,4,8,9,14       |
| 女     | 1,2,5,7,10,12,13 |
| 中性   | 11               |
| 保密   | 6                |
+--------+------------------+
```

group by + 集合函数

1. 通过 group_concat()的启发，我们既然可以统计出每个分组的某字段的值的集合，那么我们也可以通过集合函数来对这个值的集合做一些操作

```
select gender,group_concat(age) from students group by gender;
+--------+----------------------+
| gender | group_concat(age)    |
+--------+----------------------+
| 男     | 29,59,36,27,12       |
| 女     | 18,18,38,18,25,12,34 |
| 中性   | 33                   |
| 保密   | 28                   |
+--------+----------------------+
```

分别统计性别为男/女的人年龄平均值

```
select gender,avg(age) from students group by gender;
+--------+----------+
| gender | avg(age) |
+--------+----------+
| 男     |  32.6000 |
| 女     |  23.2857 |
| 中性   |  33.0000 |
| 保密   |  28.0000 |
+--------+----------+
```

分别统计性别为男/女的人的个数

```
select gender,count(*) from students group by gender;
+--------+----------+
| gender | count(*) |
+--------+----------+
| 男     |        5 |
| 女     |        7 |
| 中性   |        1 |
| 保密   |        1 |
+--------+----------+
```

group by + having

1. having 条件表达式：用来分组查询后指定一些条件来输出查询结果
2. having 作用和 where 一样，但 having 只能用于 group by

```
select gender,count(*) from students group by gender having count(*)>2;
+--------+----------+
| gender | count(*) |
+--------+----------+
| 男     |        5 |
| 女     |        7 |
+--------+----------+
```

group by + with rollup

1. with rollup 的作用是：在最后新增一行，来记录当前列里所有记录的总和，having 操作要放到 rollup 之后

```
select gender,count(*) from students group by gender with rollup;
+--------+----------+
| gender | count(*) |
+--------+----------+
| 男     |        5 |
| 女     |        7 |
| 中性   |        1 |
```

```
| 保密   |        1 |
| NULL   |       14 |
+--------+----------+


select gender,group_concat(age) from students group by gender with rollu
p;
+--------+-----------------------------------------+
| gender | group_concat(age)                       |
+--------+-----------------------------------------+
| 男     | 29,59,36,27,12                          |
| 女     | 18,18,38,18,25,12,34                    |
| 中性   | 33                                      |
| 保密   | 28                                      |
| NULL   | 29,59,36,27,12,18,18,38,18,25,12,34,33,28 |
+--------+-----------------------------------------+
```

15.6 窗口函数

-- 窗口函数（8.0 新增的）

select *,rank() over (partition by cls_id order by age desc) as rank1,

dense_rank() over (partition by cls_id order by age desc) as dese_rank,

row_number() over (partition by cls_id order by age desc) as row_num from students;

-- rank 排名是相同名次会记录数目，dense_rank 不会记录相同的

select *,rank() over (order by age desc) as rank1,

dense_rank() over (order by age desc) as dese_rank,

row_number() over (order by age desc) as row_num from students;

https://dev.mysql.com/doc/refman/8.0/en/window-function-descriptions.html
https://zhuanlan.zhihu.com/p/92654574

15.7 获取部分行（分页）

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

语法

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

说明

- 从 start 开始，获取 count 条数据，最上面一行是第 0 行

例 1：查询前 3 行男生信息

```
select * from students where gender=1 limit 0,3;
```

示例：分页

- 已知：每页显示 m 条数据，当前显示第 n 页

- 求总页数：此段逻辑后面会在 python 中实现
  - 查询总条数 p1
  - 使用 p1 除以 m 得到 p2
  - 如果整**除则 p2 为总数页**
    SELECT CEILING(count(*)/5) from student;
  - 如果不整除则 p2+1 为总页数
- 求第 n 页的数据

```
select * from students where is_delete=0 limit (n-1)*m,m
```

15.8 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

mysql 支持三种类型的连接查询，分别为：

- **内连接查询：查询的结果为两个表匹配到的数据**

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

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

语法

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

例 1：使用内连接查询班级表与学生表

```
select * from students inner join classes on students.cls_id = classes.id;
```

例 2：使用左连接查询班级表与学生表

- 此处使用了 as 为表起别名，目的是编写简单

```
select * from students as s left join classes as c on s.cls_id = c.id;
```

例 3：使用右连接查询班级表与学生表

```
select * from students as s right join classes as c on s.cls_id = c.id;
```

例 4：查询学生姓名及班级名称

```
select s.name,c.name from students as s inner join classes as c on s.cls
_id = c.id;
```


15.9 自关联

- 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

说明:

- 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

创建 areas 表的语句如下：

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

- 从 sql 文件中导入数据

```
source areas.sql;
```

- 查询一共有多少个省

```
select count(*) from areas where pid is null;
```

- 例 1：查询省的名称为“山西省”的所有城市
- `select * from areas as c inner join areas as p on c.pid=p.aid where p.atitle='山西省';`

- 例 2：查询市的名称为“广州市”的所有区县

```
select dis.* from areas as dis
inner join areas as city on city.aid=dis.pid
where city.atitle='广州市';
```


15.10 子查询

子查询

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

主查询

主要查询的对象,第一条 select 语句

主查询和子查询的关系--规则

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

子查询分类

- **标量子查询: 子查询返回的结果是一个数据(一行一列)**
- **列子查询: 返回的结果是一列(一列多行)**
- **行子查询: 返回的结果是一行(一行多列)**

标量子查询

1. 查询班级学生平均年龄
2. 查询大于平均年龄的学生

查询班级哪些学生的身高大于平均身高

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

列级子查询

- 查询还有学生在班的所有班级名字
    a. 找出学生表中所有的班级 id
    b. 找出班级表中对应的名字

```
select name from classes where id in (select cls_id from students);
```

行级子查询

- 需求: 查找班级年龄最大,身高最高的学生
- 行元素: 将多个字段合成一个行元素,在行级子查询中会使用到行元素

```
select * from students where (height,age) = (select max(height),max(age)
 from students);
```

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)

总结

查询的完整格式 ^_^ 不要被吓到 其实很简单 !_!

```
SELECT select_expr [,select_expr,...] [
      FROM tb_name
      [WHERE 条件判断]
      [GROUP BY {col_name | postion} [ASC | DESC], ...]
      [HAVING WHERE 条件判断]
      [ORDER BY {col_name|expr|postion} [ASC | DESC], ...]
      [ LIMIT {[offset,]rowcount | row_count OFFSET offset}]
]
```

- 完整的 select 语句

```
select distinct *
from 表名
```

```
where ....
group by ... having ...
order by ...
limit start,count
```

- 执行顺序为：
  - from 表名
  - where ....
  - group by ...
  - select distinct *
  - having ...
  - order by ...
  - limit start,count
- 实际使用中，只是语句中某些部分的组合，而不是全部