- 主要知识点包括：能够与 mysql 建立连接，创建数据库、表，分别从图形界面与脚本界面两个方面讲解
- 相关的知识点包括：E-R 关系模型，数据库的 3 范式，mysql 中数据字段的类型，字段约束
- 数据库的操作主要包括：
    - 数据库的操作，包括创建、删除
    - 表的操作，包括创建、修改、删除
    - 数据的操作，包括增加、修改、删除、查询，简称 crud
- 学生表结构：
    - id
    - 名称
    - 性别
    - 地址
    - 生日
- 科目表结构：
    - id
    - 名称

数据库简介

- 人类在进化的过程中，创造了数字、文字、符号等来进行数据的记录，但是承受着认知能力和创造能力的提升，数据量越来越大，对于数据的记录和准确查找，成为了一个重大难题
- 计算机诞生后，数据开始在计算机中存储并计算，并设计出了数据库系统
- 数据库系统解决的问题：持久化存储，优化读写，保证数据的有效性
- 当前使用的数据库，主要分为两类
    - 文档型，如 sqlite，就是一个文件，通过对文件的复制完成数据库的复制

    - 服务型，如 mysql、postgre，数据存储在一个物理文件中，但是需要使用终端以 tcp/ip 协议连接，进行数据库的读写操作

E-R 模型

- 当前物理的数据库都是按照 E-R 模型进行设计的
- E 表示 entry，实体
- R 表示 relationship，关系
- 一个实体转换为数据库中的一个表
- 关系描述两个实体之间的对应规则，包括
    - 一对一
    - 一对多
    - 多对多
- 关系转换为数据库表中的一个列 *在关系型数据库中一行就是一个对象

三范式

- 经过研究和对使用中问题的总结，对于设计数据库提出了一些规范，这些规范被称为范式
- 第一范式（1NF)：列不可拆分
- 第二范式（2NF)：唯一标识
- 第三范式（3NF)：引用主键
- 说明：后一个范式，都是在前一个范式的基础上建立的

安装

- 安装

```
sudo apt-get install mysql-server mysql-client
```

然后按照提示输入

管理服务

- 启动

```
service mysql start
```

- 停止

```
service mysql stop
```

- 重启

```
service mysql restart
```

允许远程连接

- 找到 mysql 配置文件并修改

```
sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf
```

将 `bind-address=127.0.0.1` 注释

- 登录 mysql，运行命令

```
grant all privileges on *.* to 'root'@'%' identified by 'mysql' with grant option;
flush privileges;
```

- 重启 mysql

数据完整性

- 一个数据库就是一个完整的业务单元，可以包含多张表，数据被存储在表中
- 在表中为了更加准确的存储数据，保证数据的正确有效，可以在创建表的时候，为表添加一些强制性的验证，包括数据字段的类型、约束

字段类型

- 在 mysql 中包含的数据类型很多，这里主要列出来常用的几种
- 数字：int,decimal
- 字符串：varchar,text
- 日期：datetime
- 布尔：bit

约束

- 主键 primary key

- 非空 not null
- 惟一 unique
- 默认 default
- 外键 foreign key

使用图形窗口连接

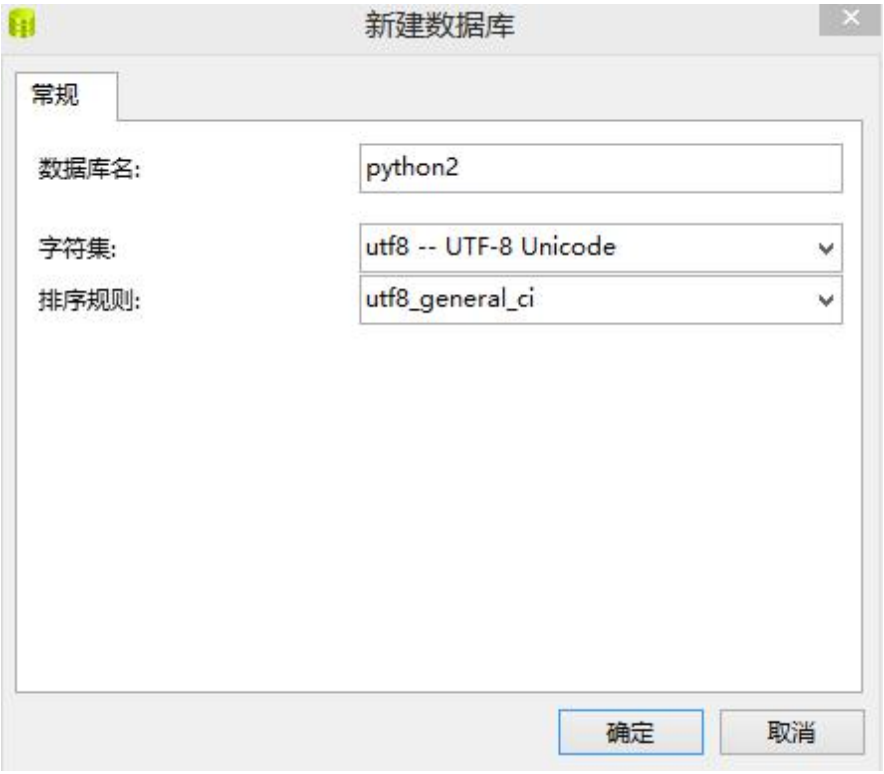
- 下发 windows 的 navicat
- 点击“连接”弹出窗口，按照提示填写连接信息，如下图

- 连接成功后，会在连接名称下面显示出当前的数据库

- 双击选中数据库，就可以编辑此数据库
- 下次再进入此软件时，通过双击完成连接、编辑操作

数据库操作

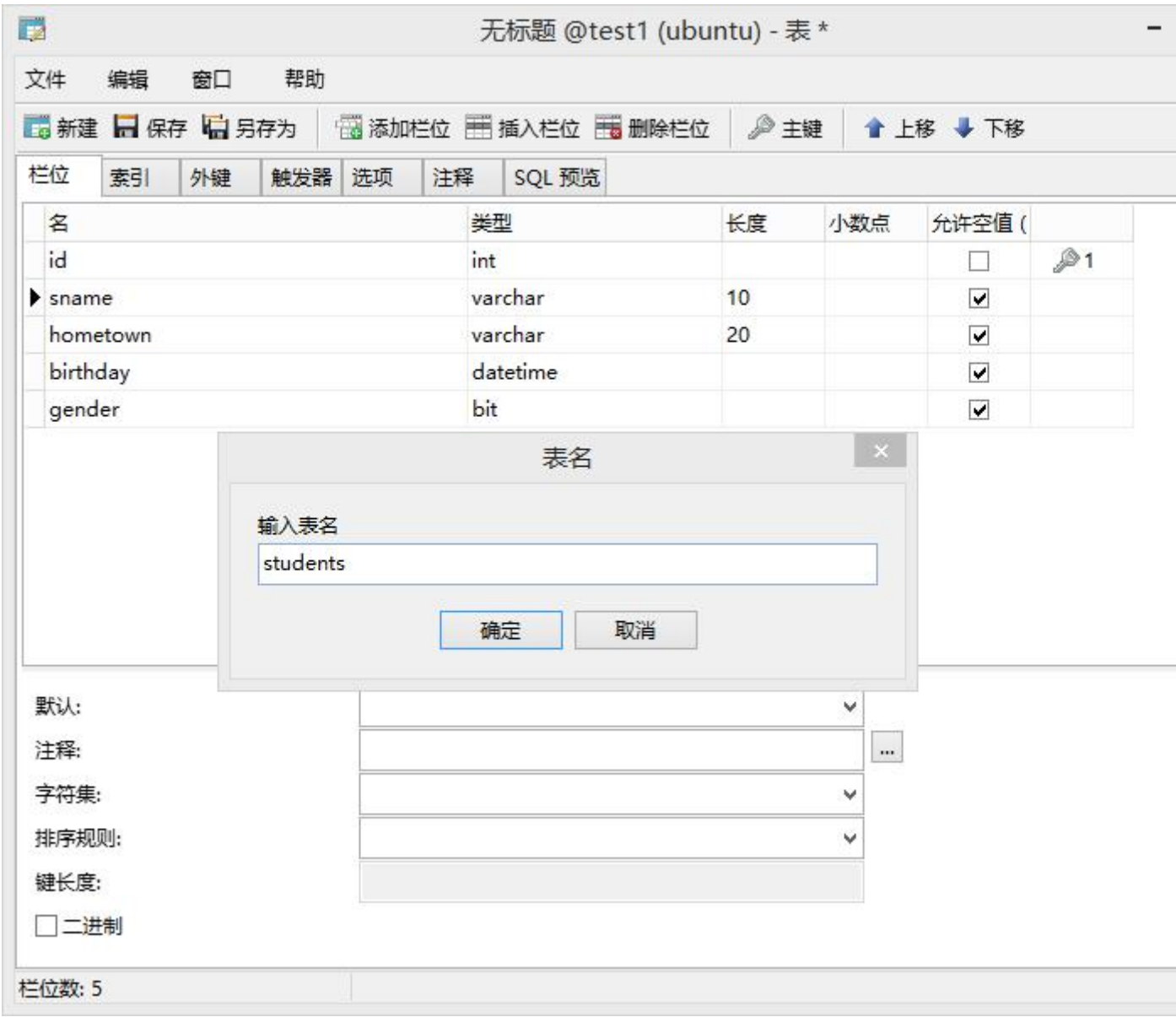
- 在连接的名称上右击，选择“新建数据库”，弹出窗口，并按提示填写



- 在数据库上右击，选择“删除数据库”可以完成删除操作

表操作

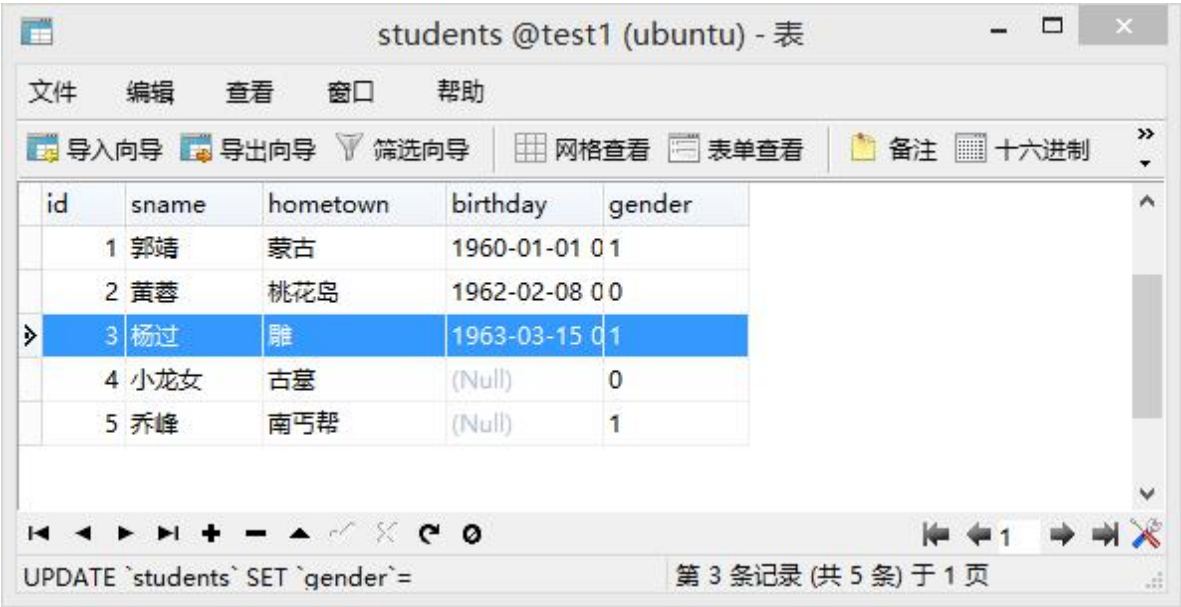
- 当数据库显示为高亮时，表示当前操作此数据库，可以在数据中创建表
- 一个实体对应一张表，用于存储特定结构的数据
- 点击“新建表”，弹出窗口，按提示填写信息



- 主键的名称一般为 id，设置为 int 型，无符号数，自动增长，非空
- 自动增长表示由 mysql 系统负责维护这个字段的值，不需要手动维护，所以不用关心这个字段的具体值
- 字符串 varchar 类型需要设置长度，即最多包含多少个字符
- 点击“添加栏位”，可以添加一个新的字段
- 点击“保存”，为表定义名称

数据操作

- 表创建成功后，可以在右侧看到，双击表打开新窗口，如下图



- 在此窗口中可以增加、修改、删除数据

逻辑删除

- 对于重要数据，并不希望物理删除，一旦删除，数据无法找回
- 一般对于重要数据，会设置一个 isDelete 的列，类型为 bit，表示逻辑删除
- 大于大量增长的非重要数据，可以进行物理删除
- 数据的重要性，要根据实际开发决定

使用命令连接

- 命令操作方式，在工作中使用的更多一些，所以要达到熟练的程度
- 打开终端，运行命令

```
mysql -uroot -p
```

回车后输入密码，当前设置的密码为 `mysql`

- 连接成功后如下图

```
python@ubuntu:~/Desktop/docMysql$ mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.13-0ubuntu0.16.04.2 (Ubuntu)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rigl

Oracle is a registered trademark of Oracle Corporation and/or it
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current in

mysql>
```

- 退出登录

`quit` 或 `exit`

- 退出成功后如下图

```
mysql> quit
Bye
python@ubuntu:~/Desktop/docMysql$
```

- 登录成功后，输入如下命令查看效果

查看版本：`select version();`
显示当前时间：`select now();`

- 注意：在语句结尾要使用分号;

远程连接

- 一般在公司开发中，可能会将数据库统一搭建在一台服务器上，所有开发人员共用一个数据库，而不是在自己的电脑中配置一个数据库
- 运行命令

```
mysql -hip 地址 -uroot -p
```

- -h 后面写要连接的主机 ip 地址
- -u 后面写连接的用户名
- -p 回车后写密码

数据库操作

- 创建数据库

```
create database 数据库名 charset=utf8;
```

- 删除数据库

```
drop database 数据库名;
```

- 切换数据库

```
use 数据库名;
```

- 查看当前选择的数据库

```
select database();
```

表操作

- 查看当前数据库中所有表

```
show tables;
```

- 创建表
- auto_increment 表示自动增长
- create table 表名(列及类型);
- 如：
- create table students(
- id int auto_increment primary key,
- sname varchar(10) not null
-);
- 修改表

```
alter table 表名 add|change|drop 列名 类型;
```

如：

```
alter table students add birthday datetime;
```

- 删除表

```
drop table 表名;
```

- 查看表结构

```
desc 表名;
```

- 更改表名称

```
rename table 原表名 to 新表名;
```

- 查看表的创建语句

```
show create table '表名';
```

数据操作

- 查询

```
select * from 表名
```

- 增加

全列插入：`insert into 表名 values(...)`
缺省插入：`insert into 表名(列1,...) values(值1,...)`
同时插入多条数据：`insert into 表名 values(...),(...)...;`
或 `insert into 表名(列1,...) values(值1,...),(值1,...)...;`

- 主键列是自动增长，但是在全列插入时需要占位，通常使用 0，插入成功后以实际数据为准

- 修改

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

- 删除

```
delete from 表名 where 条件
```

- 逻辑删除，本质就是修改操作 update

```
alter table students add isdelete bit default 0;
```

如果需要删除则

```
update students isdelete=1 where ...;
```

备份与恢复

数据备份

- 进入超级管理员

```
sudo -s
```

- 进入 mysql 库目录

```
cd /var/lib/mysql
```

- 运行 mysqldump 命令

```
mysqldump –uroot –p 数据库名 > ~/Desktop/备份文件.sql;
```

按提示输入 mysql 的密码

数据恢复

- 连接 mysqk，创建数据库
- 退出连接，执行如下命令

```
mysql -uroot –p 数据库名 < ~/Desktop/备份文件.sql
```

根据提示输入 mysql 密码

总结

- 数据库解决的问题，E-R 模型，三范式
- 图形界面操作数据库、表、数据
- 命令行操作数据库、表、数据

作业

- 完善学生表、科目表及数据
- 设计两张表，要求是一对多的关系

简介

- 查询的基本语法

```
select * from 表名;
```

- from 关键字后面写表名，表示数据来源于是这张表
- select 后面写表中的列名，如果是*表示在结果中显示表中所有列
- 在 select 后面的列名部分，可以使用 as 为列起别名，这个别名出现在结果集中
- 如果要查询多个列，之间使用逗号分隔

消除重复行

- 在 select 后面列前使用 distinct 可以消除重复的行

```
select distinct gender from students;
```

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;
```

比较运算符

- 等于=
- 大于>
- 大于等于>=
- 小于<
- 小于等于<=
- 不等于!=或<>
- 查询编号大于 3 的学生

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

- 查询编号不大于 4 的科目

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

- 查询姓名不是“黄蓉”的学生

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

- 查询没被删除的学生

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

逻辑运算符

- and
- or
- not
- 查询编号大于 3 的女同学

```
select * from students where id>3 and gender=0;
```

- 查询编号小于 4 或没被删除的学生

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

模糊查询

- like
- %表示任意多个任意字符
- _表示一个任意字符
- 查询姓黄的学生

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

- 查询姓黄并且名字是一个字的学生

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

- 查询姓黄或叫靖的学生

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

范围查询

- in 表示在一个非连续的范围内
- 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
- 查询学生是 3 至 8 的学生

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

- 查询学生是 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
- 查询没有填写地址的学生

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

- 判非空 is not null

- 查询填写了地址的学生

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

- 查询填写了地址的女生

```
select * from students where hometown is not null and gender=0;
```

优先级

- 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

分组

- 按照字段分组，表示此字段相同的数据会被放到一个组中
- 分组后，只能查询出相同的数据列，对于有差异的数据列无法出现在结果集中
- 可以对分组后的数据进行统计，做聚合运算
- 语法：

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

- 查询男女生总数

```
select gender as 性别,count(*)
from students
group by gender;
```

- 查询各城市人数

```
select hometown as 家乡,count(*)
from students
group by hometown;
```

分组后的数据筛选

- 语法：

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

- having 后面的条件运算符与 where 的相同
- 查询男生总人数

方案一

```
select count(*)
```

```
from students
where gender=1;
----------------------------------
```

方案二：

```
select gender as 性别,count(*)
from students
group by gender
having gender=1;
```

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

聚合

- 为了快速得到统计数据，提供了 5 个聚合函数
- count(*)表示计算总行数，括号中写星与列名，结果是相同的
- 查询学生总数

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

- max(列)表示求此列的最大值
- 查询女生的编号最大值

```
select max(id) from students where gender=0;
```

- min(列)表示求此列的最小值
- 查询未删除的学生最小编号

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

- sum(列)表示求此列的和
- 查询男生的编号之后

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

- avg(列)表示求此列的平均值
- 查询未删除女生的编号平均值

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

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,...
```

- 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
- 查询未删除男生学生信息，按学号降序

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

- 查询未删除科目信息，按名称升序

```
select * from subject
where isdelete=0
order by stitle;
```

总结

- 完整的 select 语句

```
select distinct *
from 表名
where ....
group by ... having ...
order by ...
limit star,count
```

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

作业

- 对学生表、科目表进行数据的查询

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
- 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

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
    - 如果不整除则 p2+1 为总页数
- 求第 n 页的数据

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

简介

- 实体与实体之间有 3 种对应关系，这些关系也需要存储下来
- 在开发中需要对存储的数据进行一些处理，用到内置的一些函数
- 视图用于完成查询语句的封装
- 事务可以保证复杂的增删改操作有效

先看个问题

- 问：查询每个学生每个科目的分数
- 分析：学生姓名来源于students表，科目名称来源于subjects，分数来源于scores表，怎么将 3 个表放到一起查询，并将结果显示在同一个结果集中呢？
- 答：当查询结果来源于多张表时，需要使用连接查询
- 关键：找到表间的关系，当前的关系是
    - students 表的 id---scores 表的 stuid
    - subjects 表的 id---scores 表的 subid
- 则上面问题的答案是：

```
select students.sname,subjects.stitle,scores.score
from scores
inner join students on scores.stuid=students.id
inner join subjects on scores.subid=subjects.id;
```

- 结论：当需要对有关系的多张表进行查询时，需要使用连接 join

连接查询

- 连接查询分类如下：
    - 表 A inner join 表 B：表 A 与表 B 匹配的行会出现在结果中
    - 表 A left join 表 B：表 A 与表 B 匹配的行会出现在结果中，外加表 A 中独有的数据，未对应的数据使用 null 填充
    - 表 A right join 表 B：表 A 与表 B 匹配的行会出现在结果中，外加表 B 中独有的数据，未对应的数据使用 null 填充
- 在查询或条件中推荐使用“表名.列名”的语法
- 如果多个表中列名不重复可以省略“表名.”部分
- 如果表的名称太长，可以在表名后面使用' as 简写名'或' 简写名'，为表起个临时的简写名称

练习

- 查询学生的姓名、平均分

```
select students.sname,avg(scores.score)
```

```
from scores
inner join students on scores.stuid=students.id
group by students.sname;
```

- 查询男生的姓名、总分

```
select students.sname,avg(scores.score)
from scores
inner join students on scores.stuid=students.id
where students.gender=1
group by students.sname;
```

- 查询科目的名称、平均分

```
select subjects.stitle,avg(scores.score)
from scores
inner join subjects on scores.subid=subjects.id
group by subjects.stitle;
```

- 查询未删除科目的名称、最高分、平均分

```
select subjects.stitle,avg(scores.score),max(scores.score)
from scores
inner join subjects on scores.subid=subjects.id
where subjects.isdelete=0
group by subjects.stitle;
```

字符串函数

- 查看字符的 ascii 码值 ascii(str)，str 是空串时返回 0

```
select ascii('a');
```

- 查看 ascii 码值对应的字符 char(数字)

```
select char(97);
```

- 拼接字符串 concat(str1,str2...)

```
select concat(12,34,'ab');
```

- 包含字符个数 length(str)

```
select length('abc');
```

- 截取字符串
    - left(str,len)返回字符串 str 的左端 len 个字符
    - right(str,len)返回字符串 str 的右端 len 个字符
    - substring(str,pos,len)返回字符串 str 的位置 pos 起 len 个字符

```
select substring('abc123',2,3);
```

- 去除空格
    - ltrim(str)返回删除了左空格的字符串 str
    - rtrim(str)返回删除了右空格的字符串 str

    - trim([方向 remstr from str)返回从某侧删除 remstr 后的字符串 str，方向词包括 both、leading、trailing，表示两侧、左、右

```
select trim('  bar   ');
select trim(leading 'x' FROM 'xxxbarxxx');
select trim(both 'x' FROM 'xxxbarxxx');
select trim(trailing 'x' FROM 'xxxbarxxx');
```

- 返回由 n 个空格字符组成的一个字符串 space(n)

```
select space(10);
```

- 替换字符串 replace(str,from_str,to_str)

```
select replace('abc123','123','def');
```

- 大小写转换，函数如下
    - lower(str)
    - upper(str)

```
select lower('aBcD');
```

数学函数

- 求绝对值 abs(n)

```
select abs(-32);
```

- 求 m 除以 n 的余数 mod(m,n)，同运算符%

```
select mod(10,3);
select 10%3;
```

- 地板 floor(n)，表示不大于 n 的最大整数

```
select floor(2.3);
```

- 天花板 ceiling(n)，表示不小于 n 的最大整数

```
select ceiling(2.3);
```

- 求四舍五入值 round(n,d)，n 表示原数，d 表示小数位置，默认为 0

```
select round(1.6);
```

- 求 x 的 y 次幂 pow(x,y)

```
select pow(2,3);
```

- 获取圆周率 PI()

```
select PI();
```

- 随机数 rand()，值为 0-1.0 的浮点数

```
select rand();
```

- 还有其它很多三角函数，使用时可以查询文档

日期时间函数

- 获取子值，语法如下
    - year(date)返回 date 的年份(范围在 1000 到 9999)
    - month(date)返回 date 中的月份数值
    - day(date)返回 date 中的日期数值
    - hour(time)返回 time 的小时数(范围是 0 到 23)
    - minute(time)返回 time 的分钟数(范围是 0 到 59)
    - second(time)返回 time 的秒数(范围是 0 到 59)

```
select year('2016-12-21');
```

- 日期计算，使用+-运算符，数字后面的关键字为 year、month、day、hour、minute、second

```
select '2016-12-21'+interval 1 day;
```

- 日期格式化 date_format(date,format)，format 参数可用的值如下
    - 获取年%Y，返回 4 位的整数
        * 获取年%y，返回 2 位的整数
        * 获取月%m，值为 1-12 的整数
    - 获取日%d，返回整数
        * 获取时%H，值为 0-23 的整数
        * 获取时%h，值为 1-12 的整数
        * 获取分%i，值为 0-59 的整数
        * 获取秒%s，值为 0-59 的整数

```
select date_format('2016-12-21','%Y %m %d');
```

- 当前日期 current_date()

```
select current_date();
```

- 当前时间 current_time()

```
select current_time();
```

- 当前日期时间 now()

```
select now();
```

视图

- 对于复杂的查询，在多次使用后，维护是一件非常麻烦的事情
- 解决：定义视图
- 视图本质就是对查询的一个封装
- 定义视图

```
create view stuscore as
select students.*,scores.score from scores
inner join students on scores.stuid=students.id;
```

- 视图的用途就是查询

```
select * from stuscore;
```

子查询

- 查询支持嵌套使用
- 查询各学生的语文、数学、英语的成绩

```
select sname,
(select sco.score from scores sco inner join subjects sub on sco.subid=sub.id
where sub.stitle='语文' and stuid=stu.id) as 语文,
(select sco.score from  scores sco inner join subjects sub on
sco.subid=sub.id where sub.stitle='数学' and stuid=stu.id) as 数学,
(select sco.score from  scores sco inner join subjects sub on
sco.subid=sub.id where sub.stitle='英语' and stuid=stu.id) as 英语
from students stu;
```

事务

- 当一个业务逻辑需要多个 sql 完成时，如果其中某条 sql 语句出错，则希望整个操作都退回
- 使用事务可以完成退回的功能，保证业务逻辑的正确性
- 事务四大特性(简称 ACID)
    - 原子性(Atomicity)：事务中的全部操作在数据库中是不可分割的，要么全部完成，要么均不执行
    - 一致性(Consistency)：几个并行执行的事务，其执行结果必须与按某一顺序串行执行的结果相一致

    - 隔离性(Isolation)：事务的执行不受其他事务的干扰，事务执行的中间结果对其他事务必须是透明的
    - 持久性(Durability)：对于任意已提交事务，系统必须保证该事务对数据库的改变不被丢失，即使数据库出现故障

- 要求：表的类型必须是 innodb 或 bdb 类型，才可以对此表使用事务
- 查看表的创建语句

```
show create table students;
```

- 修改表的类型

```
alter table '表名' engine=innodb;
```

- 事务语句

开启 `begin;`
提交 `commit;`
回滚 `rollback;`

示例 1

- 步骤 1：打开两个终端，连接 mysql，使用同一个数据库，操作同一张表

终端 1：

```
select * from students;
```

终端 2：

```
begin;
insert into students(sname) values('张飞');
```

- 步骤 2

终端 1：

```
select * from students;
```

- 步骤 3

终端 2：

```
commit;
```

终端 1：

```
select * from students;
```

示例 2

- 步骤 1：打开两个终端，连接 mysql，使用同一个数据库，操作同一张表

终端 1：

```
select * from students;
```

```
------------------------
```
终端 2：
```
begin;
insert into students(sname) values('张飞');
```

- 步骤 2

终端 1：
```
select * from students;
```

- 步骤 3

终端 2：
```
rollback;
------------------------
```
终端 1：
```
select * from students;
```

总结

- 关系的存储
- 连接查询
- 自关联
- 子查询
- 常用内置函数
- 视图
- 事务

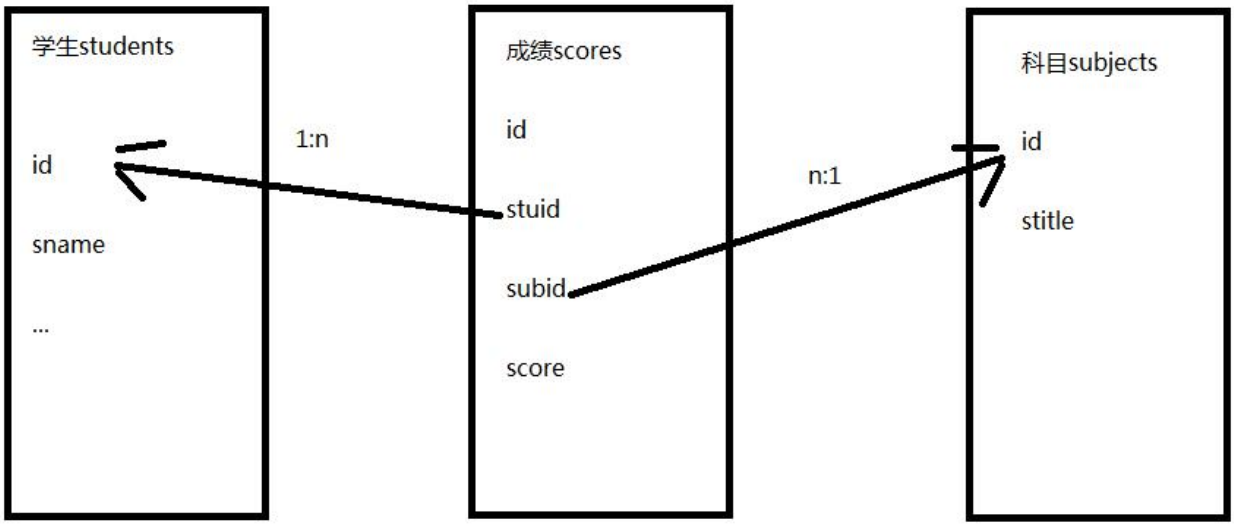
作业

- 设计班级表，与学生表关联，并进行查询
- 设计分类表，自关联，并进行查询
- 创建视图存储上面的两个查询

关系

- 创建成绩表 scores，结构如下
    - id

    - 学生
    - 科目
    - 成绩
-
- 思考：学生列应该存什么信息呢？
- 答：学生列的数据不是在这里新建的，而应该从学生表引用过来，关系也是一条数据；根据范式要求应该存储学生的编号，而不是学生的姓名等其它信息
- 同理，科目表也是关系列，引用科目表中的数据



- 创建表的语句如下

```
create table scores(
id int primary key auto_increment,
stuid int,
subid int,
score decimal(5,2)
);
```

外键

- 思考：怎么保证关系列数据的有效性呢？任何整数都可以吗？
- 答：必须是学生表中 id 列存在的数据，可以通过外键约束进行数据的有效性验证
- 为 stuid 添加外键约束

```
alter table scores add constraint stu_sco foreign key(stuid) references
students(id);
```

- 此时插入或者修改数据时，如果 stuid 的值在 students 表中不存在则会报错
- 在创建表时可以直接创建约束

```
create table scores(
id int primary key auto_increment,
stuid int,
subid int,
score decimal(5,2),
foreign key(stuid) references students(id),
foreign key(subid) references subjects(id)
);
```

外键的级联操作

- 在删除 students 表的数据时，如果这个 id 值在 scores 中已经存在，则会抛异常
- 推荐使用逻辑删除，还可以解决这个问题
- 可以创建表时指定级联操作，也可以在创建表后再修改外键的级联操作
- 语法

```
alter table scores add constraint stu_sco foreign key(stuid) references
students(id) on delete cascade;
```

- 级联操作的类型包括：
    - restrict（限制）：默认值，抛异常
    - cascade（级联）：如果主表的记录删掉，则从表中相关联的记录都将被删除
    - set null：将外键设置为空
    - no action：什么都不做

与 python 交互

- 在熟练使用 sql 语句的基础上，开始使用 python 语言提供的模块与 mysql 进行交互
- 这是我们在工作中大事要做的事
- 先学会 sql 是基础，一定要熟练编写 sql 语句

安装引入模块

- 安装 mysql 模块

```
sudo apt-get install python-mysql
```

- 在文件中引入模块

```
import Mysqldb
```

Connection 对象

- 用于建立与数据库的连接
- 创建对象：调用 connect()方法

```
conn=connect(参数列表)
```

- 参数 host：连接的 mysql 主机，如果本机是'localhost'
- 参数 port：连接的 mysql 主机的端口，默认是 3306
- 参数 db：数据库的名称
- 参数 user：连接的用户名
- 参数 password：连接的密码
- 参数 charset：通信采用的编码方式，默认是'gb2312'，要求与数据库创建时指定的编码一致，否则中文会乱码

对象的方法

- close()关闭连接
- commit()事务，所以需要提交才会生效
- rollback()事务，放弃之前的操作
- cursor()返回 Cursor 对象，用于执行 sql 语句并获得结果

Cursor 对象

- 执行 sql 语句
- 创建对象：调用 Connection 对象的 cursor()方法

```
cursor1=conn.cursor()
```

对象的方法

- close()关闭
- execute(operation [, parameters])执行语句，返回受影响的行数

- fetchone()执行查询语句时，获取查询结果集的第一个行数据，返回一个元组
- next()执行查询语句时，获取当前行的下一行
- fetchall()执行查询时，获取结果集的所有行，一行构成一个元组，再将这些元组装入一个元组返回
- scroll(value[,mode])将行指针移动到某个位置
    - mode 表示移动的方式
    - mode 的默认值为 relative，表示基于当前行移动到 value，value 为正则向下移动，value 为负则向上移动
    - mode 的值为 absolute，表示基于第一条数据的位置，第一条数据的位置为 0

对象的属性

- rowcount 只读属性，表示最近一次 execute()执行后受影响的行数
- connection 获得当前连接对象

增加

- 创建 testInsert.py 文件，向学生表中插入一条数据

```
#encoding=utf-8
import MySQLdb
try:

conn=MySQLdb.connect(host='localhost',port=3306,db='test1',user='root',
passwd='mysql',charset='utf8')
    cs1=conn.cursor()
    count=cs1.execute("insert into students(sname) values('张良')")
    print count
    conn.commit()
    cs1.close()
    conn.close()
except Exception,e:
    print e.message
```

修改

- 创建 testUpdate.py 文件，修改学生表的一条数据

```
#encoding=utf-8
import MySQLdb
try:
```

```
conn=MySQLdb.connect(host='localhost',port=3306,db='test1',user='root',
passwd='mysql',charset='utf8')
    cs1=conn.cursor()
    count=cs1.execute("update students set sname='刘邦' where id=6")
    print count
    conn.commit()
    cs1.close()
    conn.close()
except Exception,e:
    print e.message
```

删除

- 创建 testDelete.py 文件，删除学生表的一条数据

```
#encoding=utf-8
import MySQLdb
try:

conn=MySQLdb.connect(host='localhost',port=3306,db='test1',user='root',
passwd='mysql',charset='utf8')
    cs1=conn.cursor()
    count=cs1.execute("delete from students where id=6")
    print count
    conn.commit()
    cs1.close()
    conn.close()
except Exception,e:
    print e.message
```

sql 语句参数化

- 创建 testInsertParam.py 文件，向学生表中插入一条数据

```
#encoding=utf-8
import MySQLdb
try:

conn=MySQLdb.connect(host='localhost',port=3306,db='test1',user='root',
passwd='mysql',charset='utf8')
    cs1=conn.cursor()
    sname=raw_input("请输入学生姓名：")
    params=[sname]
    count=cs1.execute('insert into students(sname) values(%s)',params)
    print count
    conn.commit()
    cs1.close()
    conn.close()
except Exception,e:
    print e.message
```

其它语句

- cursor 对象的 execute()方法，也可以用于执行 create table 等语句
- 建议在开发之初，就创建好数据库表结构，不要在这里执行

查询一行数据

- 创建 testSelectOne.py 文件，查询一条学生信息

```
#encoding=utf8
import MySQLdb
try:

conn=MySQLdb.connect(host='localhost',port=3306,db='test1',user='root',
passwd='mysql',charset='utf8')
    cur=conn.cursor()
    cur.execute('select * from students where id=7')
    result=cur.fetchone()
    print result
    cur.close()
    conn.close()
except Exception,e:
    print e.message
```

查询多行数据

- 创建 testSelectMany.py 文件，查询一条学生信息

```
#encoding=utf8
import MySQLdb
try:

conn=MySQLdb.connect(host='localhost',port=3306,db='test1',user='root',
passwd='mysql',charset='utf8')
    cur=conn.cursor()
    cur.execute('select * from students')
    result=cur.fetchall()
    print result
    cur.close()
    conn.close()
except Exception,e:
    print e.message
```

封装

- 观察前面的文件发现，除了 sql 语句及参数不同，其它语句都是一样的
- 创建 MysqlHelper.py 文件，定义类

```
#encoding=utf8
import MySQLdb

class MysqlHelper():
    def __init__(self,host,port,db,user,passwd,charset='utf8'):
```

```
        self.host=host
        self.port=port
        self.db=db
        self.user=user
        self.passwd=passwd
        self.charset=charset

    def connect(self):

self.conn=MySQLdb.connect(host=self.host,port=self.port,db=self.db,user
=self.user,passwd=self.passwd,charset=self.charset)
        self.cursor=self.conn.cursor()

    def close(self):
        self.cursor.close()
        self.conn.close()

    def get_one(self,sql,params=()):
        result=None
        try:
            self.connect()
            self.cursor.execute(sql, params)
            result = self.cursor.fetchone()
            self.close()
        except Exception, e:
            print e.message
        return result

    def get_all(self,sql,params=()):
        list=()
        try:
            self.connect()
            self.cursor.execute(sql,params)
            list=self.cursor.fetchall()
            self.close()
        except Exception,e:
            print e.message
        return list

    def insert(self,sql,params=()):
        return self.__edit(sql,params)

    def update(self, sql, params=()):
        return self.__edit(sql, params)

    def delete(self, sql, params=()):
        return self.__edit(sql, params)

    def __edit(self,sql,params):
        count=0
        try:
            self.connect()
            count=self.cursor.execute(sql,params)
            self.conn.commit()
            self.close()
        except Exception,e:
            print e.message
```

```
        return count
```

添加

- 创建 testInsertWrap.py 文件，使用封装好的帮助类完成插入操作

```
#encoding=utf8
from MysqlHelper import *

sql='insert into students(sname,gender) values(%s,%s)'
sname=raw_input("请输入用户名：")
gender=raw_input("请输入性别，1 为男，0 为女")
params=[sname,bool(gender)]

mysqlHelper=MysqlHelper('localhost',3306,'test1','root','mysql')
count=mysqlHelper.insert(sql,params)
if count==1:
    print 'ok'
else:
    print 'error'
```

查询一个

- 创建 testGetOneWrap.py 文件，使用封装好的帮助类完成查询最新一行数据操作

```
#encoding=utf8
from MysqlHelper import *

sql='select sname,gender from students order by id desc'

helper=MysqlHelper('localhost',3306,'test1','root','mysql')
one=helper.get_one(sql)
print one
```

实例：用户登录

创建用户表 userinfos

- 表结构如下
    - id
    - uname
    - upwd
    - isdelete
- 注意：需要对密码进行加密
- 如果使用 md5 加密，则密码包含 32 个字符

- 如果使用 sha1 加密，则密码包含 40 个字符，推荐使用这种方式

```
create table userinfos(
id int primary key auto_increment,
uname varchar(20),
upwd char(40),
isdelete bit default 0
);
```

加入测试数据

- 插入如下数据，用户名为 123,密码为 123,这是 sha1 加密后的值

```
insert into userinfos
values(0,'123','40bd001563085fc35165329ea1ff5c5ecbdbbeef',0);
```

接收输入并验证

- 创建 testLogin.py 文件，引入 hashlib 模块、MysqlHelper 模块
- 接收输入
- 根据用户名查询，如果未查到则提示用户名不存在
- 如果查到则匹配密码是否相等，如果相等则提示登录成功
- 如果不相等则提示密码错误

```
#encoding=utf-8
from MysqlHelper import MysqlHelper
from hashlib import sha1

sname=raw_input("请输入用户名：")
spwd=raw_input("请输入密码:")

s1=sha1()
s1.update(spwd)
spwdSha1=s1.hexdigest()

sql="select upwd from userinfos where uname=%s"
params=[sname]

sqlhelper=MysqlHelper('localhost',3306,'test1','root','mysql')
userinfo=sqlhelper.get_one(sql,params)
if userinfo==None:
    print '用户名错误'
elif userinfo[0]==spwdSha1:
    print '登录成功'
else:
    print '密码错误'
```

总结

- python 操作数据库的类型及主要成员
- 使用 python 类完成 crud 操作
- 封装操作数据库的工具类

作业

- 封装数据库操作工具类
- 使用工具类完成 crud 操作
- 独立实现用户登录
- 实现用户注册