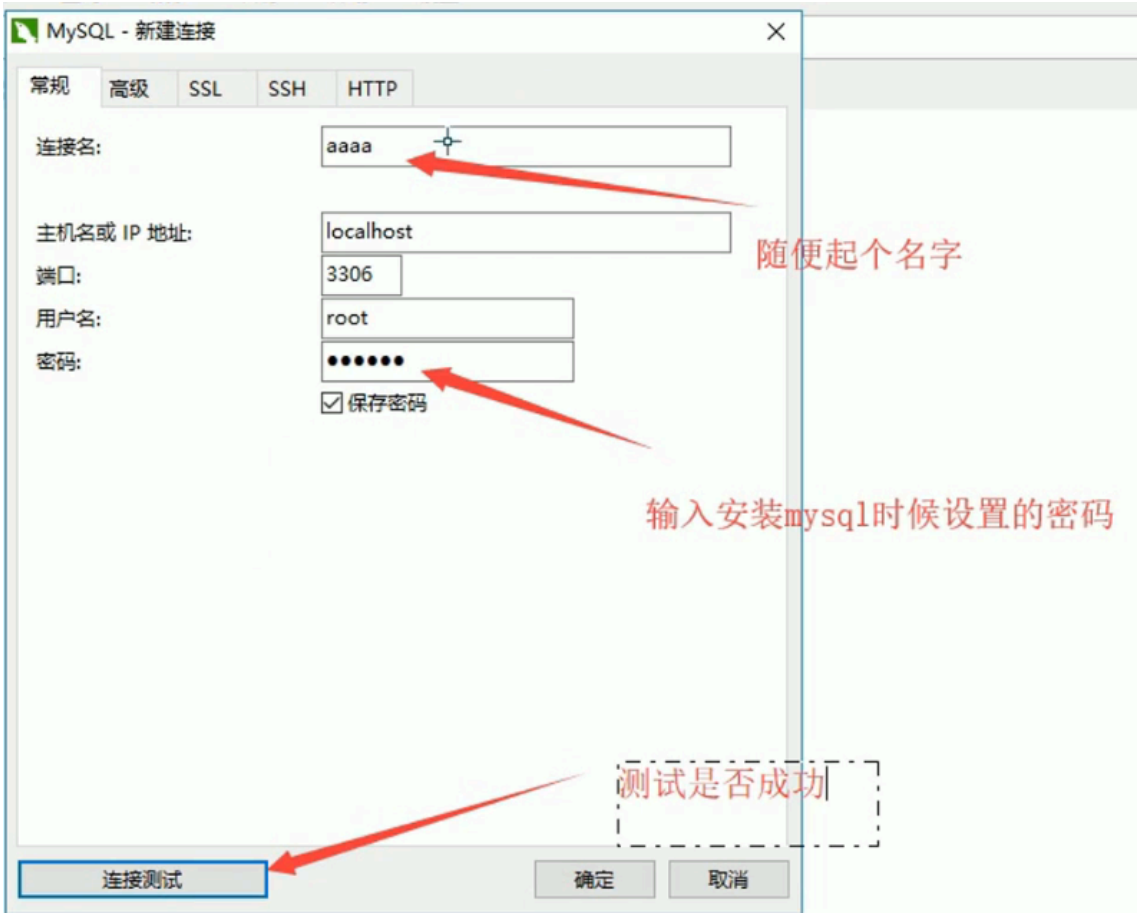
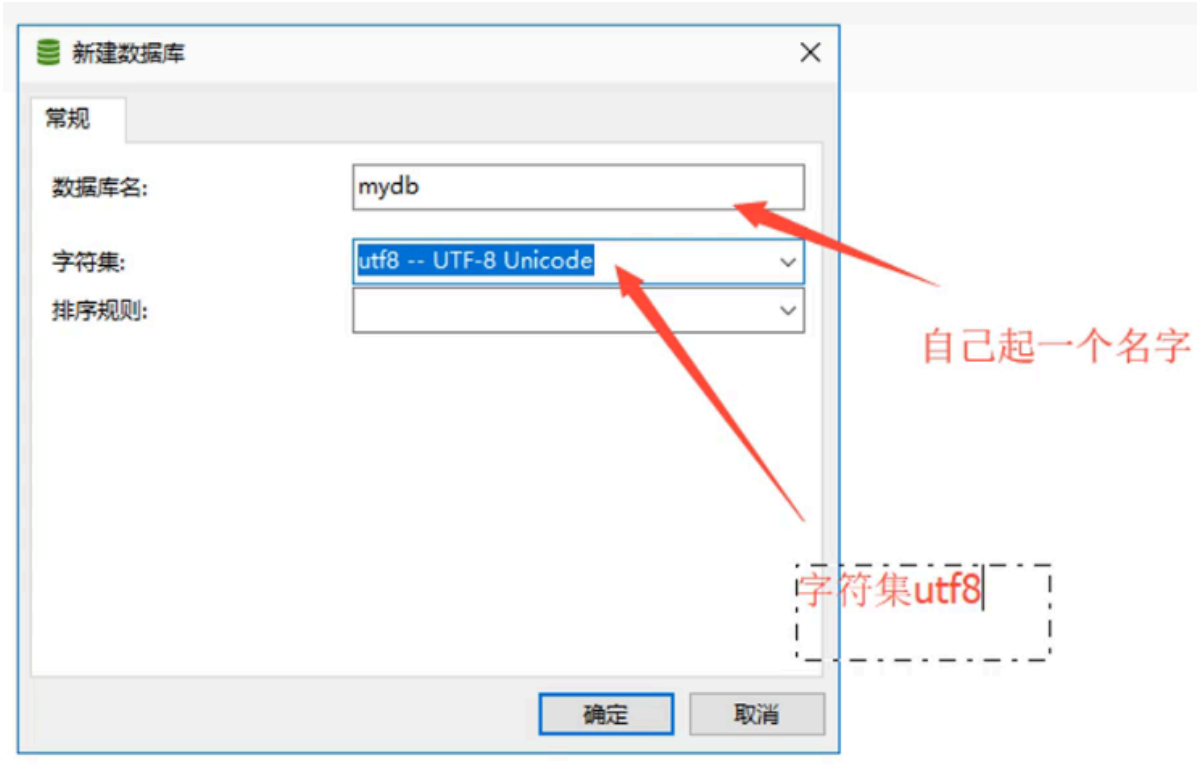
1.Navicat基本使用

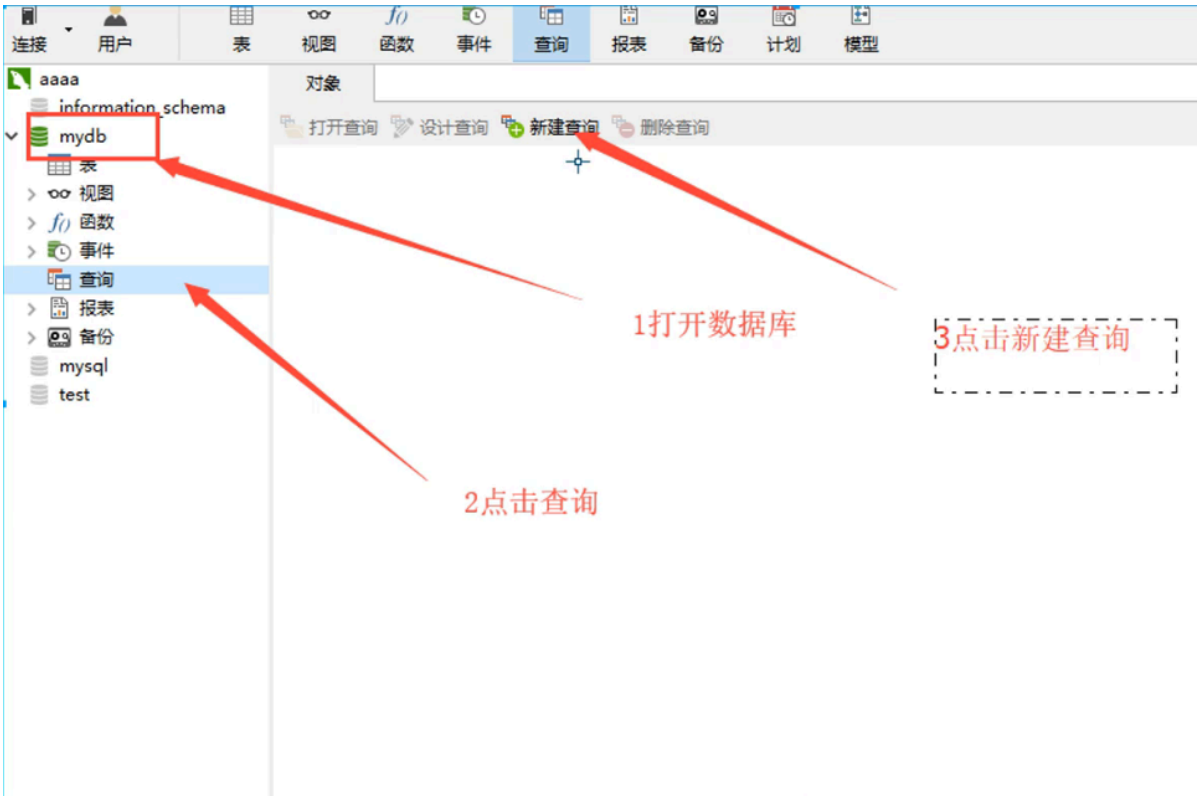
- 连接mysql

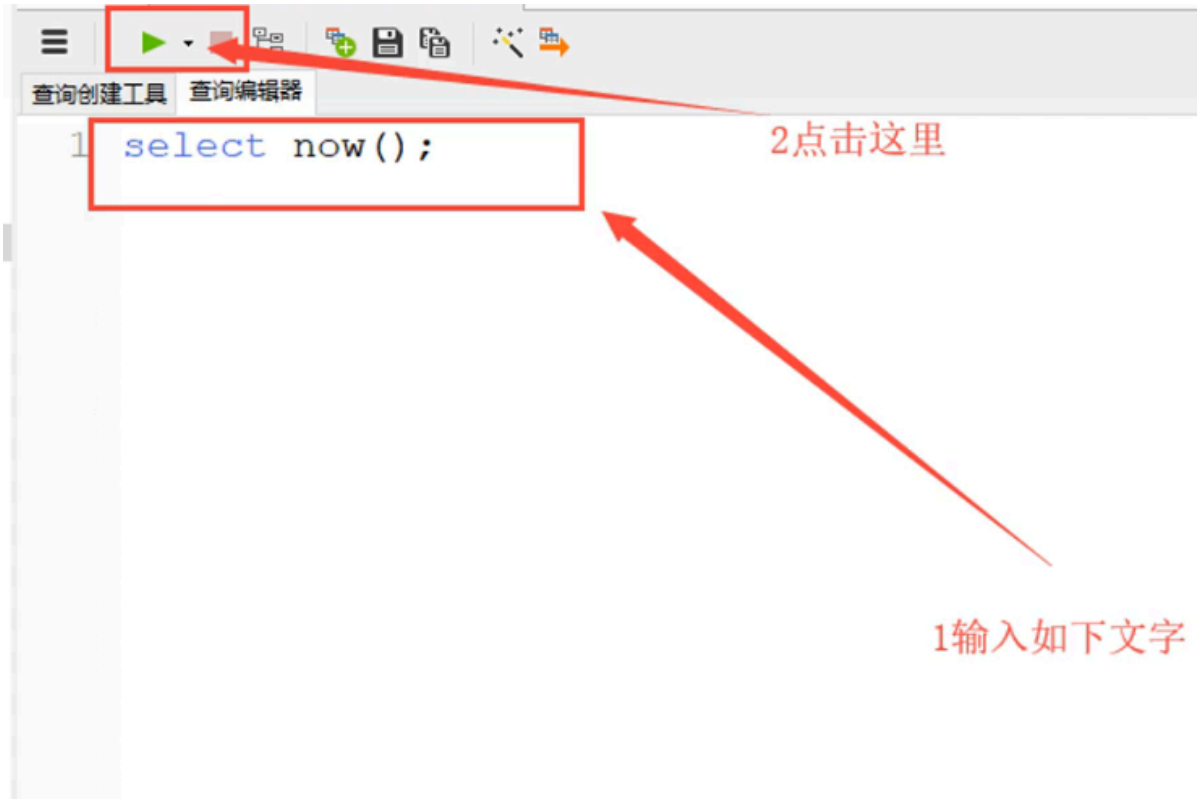


- 连接到mysql之后，首先要创建数据库
- 用鼠标右键选择新建数据库

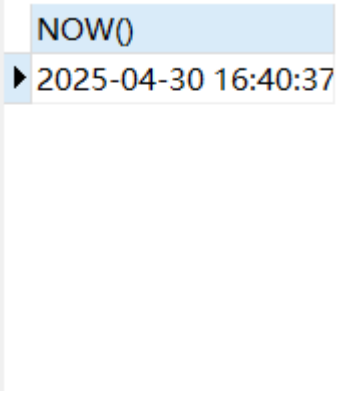


- 打开数据库，新建查询





- 看到如下结果，代表navicat连接mysql设置成功



2.sql语言中的注释

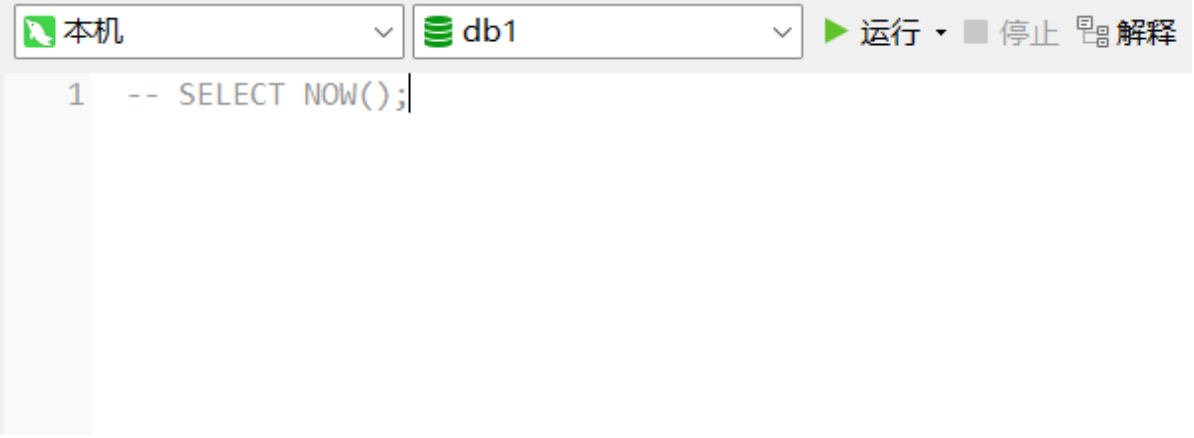
- 多行注释

格式：/*注释内容*/

```
/*
这是是一个
多行注释的例子
*/
```

- 在 Navicat 中按 ctrl+/ 快速注释选中的 SQL 代码
- 在 Navicat 中按 ctr+shiftl+/ 选中 SQL 代码取消注释

- 看到如下结果，则说明该行被注释



3.mysql的常用数据类型

- 整数：int，有符号范围（-2147483648 ,2147483647），无符号范围（0，4294967295），如：int unsigned，代表设置一个无符号的整数;
- 小整数：tinyint， 有符号范围(-128,127)，无符号范围 (0,255)，如：tinyint unsigned，代表设置一个无符号的小整数
- 小数：decimal，如 decimal(5,2)表示共存 5 位数，小数占 2 位，不能超过 2 位；整数占 3 位，不能超过三位;
- 字符串：varchar，如 varchar(3)表示最多存 3 个字符，一个中文或一个字母都占一个字符;
- 日期时间: datetime，范围（1000-01-01 00:00:00 ~ 9999-12-31 23:59:59），如'2020-01-01 12:29:59'。

4.数据库中的元素

- 数据库----database
- 表----table
- 字段(列)----field
- 记录(行)----record

5.创建表

- 语法 create table 表名（字段名 字段类型, 字段名 字段类型）；

```
-- 例1：创建表a，字段要求：name(姓名)，数据类型；varchar（字符串），长度为10
CREATE TABLE a(name VARCHAR(10));
```

- 创建两个字段的表

```
CREATE TABLE b(
  name varchar(10),
   height decimal(5,2)
);
```

- 创建三个字段的表

```
CREATE TABLE c(
   id int,
   name varchar(20),
      age tinyint unsigned
);
```

6.插入数据

- 语法 insert into 表名 values(数值, 数值, 数值);

```
INSERT INTO c VALUES('0', '张三', '20' );
```

- 指定字段插入，语法：insert into 表名(id, name) values(值, 值);

```
INSERT INTO c (id, name) VALUES (3, '曹操');
```

7.插入多条记录

- 多条insert语句，用分号隔开

```
INSERT INTO c VALUES(5, '周瑜', 50);
INSERT INTO c(id, name) VALUES (6, '鲁肃');
INSERT INTO c (name) values('诸葛亮');
```

- 一条insert插入多条记录
- 语法：insert into 表名 values(值, 值), (值, 值), (值, 值);

```
INSERT INTO c
VALUES
    ( 11, '张三', 10 ),
    ( 12, '李四', 25 ),
    ( 13, '王五', 30 );
```

- 一条insert指定字段插入多条记录
- 语法：insert into 表名(字段名, 字段名) values(值, 值), (值, 值), (值, 值);

```
INSERT INTO c(id, name)
VALUES
    ( 30, '库里'),
    ( 37, '杜兰特'),
    ( 23, '詹姆斯');
```

8.select查询表

- 语法：select*from 表名;

查询所有字段

```
SELECT *FROM c;
```

指定字段名查询

- 语法：select 字段名，字段名 from 表名;

```
-- 查询表c的id字段
select id from c;
-- 查询表c的id和age字段
select id, age from c;
-- 查询表c的所有字段，但顺序自定义
select name, id, age from c;
```

9.update修改数据

- 语法：update 表名 set 字段=值，字段=值 where 条件
    - 如果没有where条件代表修改表中的所有的记录

```
-- 例1：修改表 c，所有人的年龄（age 字段）改为50
update c set age = 50;
```

- 带有条件的update语句

```sql
-- 例2：修改表 c，
-- id 为 3 的记录，
-- 姓名（name 字段）改为'狄仁杰'，年龄（age 字段）改为 20
update c set name='狄仁杰', age = 20 where id = 3;
```

- 另外的例子

```sql
-- 修改name为刘备的记录为李白
update c set name = '李白' where name ='刘备';
```

```sql
-- id大于10的记录，长一岁
update c set age = age + 1 where id >10;
```

10.delete删除记录

- 语法：delete from 表名 where 条件

```sql
-- 例1：删除表c中id为6的记录
DELETE from c where id = 6;
```

- 另外的例子

```sql
-- 例1：删除表c中name为诸葛亮的记录
DELETE from c where name = '诸葛亮';

-- 删除年龄大于50的记录
DELETE from c where age > 50;

-- 删除id小于3的记录
delete from c where id < 3;
```

11.truncate table删除表的数据

- truncate table 表名

```sql
-- 删除表c中的所有记录
truncate table c;
```

十一、DELETE 和 TRUNCATE 的区别

- 在速度上，truncate > delete;
- 如果想删除部分数据用delete，注意带上 where子句;
- 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;

12.增删改查小结

- 增
  - insert
- 删
  - delete
- 改
  - update
- 查
  - select

13.删除表

- 语法一：drop table 表名

```
-- 删除表a
drop table a;
```

- 语法二：drop table if exists 表名

```
-- 如果表a存在，就删除表a，如果不存在，什么也不做
DROP table if exists a;

-- 如果表b存在，就删除表b，如果不存在，什么也不做
DROP table if exists b;
```

14.字段的约束

一、常用约束介绍

- 主键(primary key)：值不能重复，auto_increment 代表值自动增长；
- 非空(not null)：此字段不允许填写空值；
- 惟一(unique)：此字段的值不允许重复；
- 默认值(default)：当不填写此值时会使用默认值，如果填写时以填写为准。

二、创建带约束字段的语法格式

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

15.主键

- 主键的值不能重复
- 自增长，auto_increment
    - 值会系统自动维护，自动增长

```sql
-- 例1：创建表d，字段要求如下：
-- id：数据类型为 int unsigned（无符号整数），primary key（主键），auto_increment（自增长）
-- name 姓名：数据类型为 varchar（字符串）为10
-- age 年龄：数据类型为 int（整数）

create table d(
id int unsigned PRIMARY key auto_incremnet,
name varchar(20),
age int
);

-- 插入的时候指定了id的值
insert into d(id, name, age) values(6, '曹操', 30);

-- 不指定id的值
INSERT into d(name. age) values('周瑜', 30);
select *from d;
```

```sql
-- 如果不指定字段，主键自增长字段的值可以用占位符，0或者null
INSERT into d values(0, '天', 30);
INSERT into d values(NULL, '地', 50);
```

16.非空

- 非空not null
    - 这个字段必须有值，如果没有值，insert插入会失败

```sql
create table e(
  id int UNSIGNED,
   name varchar(10) not null,
   age int);

insert into e values(1, '张三', 20);
insert into e(id, age) values(1, 20);
select *from e;
```

17.唯一

- 唯一unique
    - 字段的约束为唯一，表示字段的值不能重复

```sql
create table 表名(
 字段名 数据类型 unique,
   ......
);
```

18.默认值

- default值
- 当一个字段没有默认值约束，插入数据时，如果指定了值，那么默认值无效，如果没有指定值，会使用默认值

```sql
create table 表名(
  字段名 数据类型 default 值,
   ...
);
```

19.字段的别名

- 通过字段名 as 别名 的语法，可以给字段起一个别名，别名可以是中文
- as可以省略
- 字段名 as 别名 和字段名 别名 结果是一样的

```sql
-- 通过as 给字段起一个别名
select card as 身份证, name as 姓名,sex as 性别 from students;

-- 别名的as可以省略
select card 身份证, name 姓名,sex 性别 from students;
```

20.表的别名

- 通过 表名 as 别名 给表起一个别名
- as可以省略

```sql
-- 通过as 给表students起一个别名
select * from students as stu;

-- 可以省略as
select * from students stu;
```

21.distinct过滤重复记录

- 通过select distinct 字段名，字段名 from 表名 来过滤select查询结果中的重复记录

```sql
select distinct sex, class from students;
```

22.where子句

- where后面跟一个条件，实现有选择的查询
- select*from 表名 where 条件

```sql
select *from students where studentNo = '001';
```

```sql
select name, class from students where age = 30;
```

23.select查询的基本规律

- select*或者select 字段名 控制了查询返回什么样的字段（列）
- where条件 控制了查询返回什么样的记录（行）

24.比较运算符

- < 小于
- <= 小于等于
- > 大于
- >= 大于等于
- !=和<>不等于

25.逻辑运算符

- and与
  - 条件1 and 条件2
  - 两个条件必须都满足
- or或
  - 条件1or条件2
  - 两个条件只要满足一个即可
- not非
  - not条件按
  - 条件成立，not以后就不成立，条件不成立，not以后就成立

26.模糊查询

- like实现模糊查询
- %代表任意多个字符
- _代表任意一个字符
- 字段名 like '字符%'
  - 指定字符开始，后面任意多个字符

```
select *from students where name like"孙%";

select *from studets where name like'孙_';
```

27.范围查找

- in（值，值，值）
  - 非连续范围查找
- between 开始值 and 结束值
  - 连续范围查找，包含开始值，包含结束值

```sql
SELECT *from students where hometown ='北京' or hometown ='上海' or hometown = '广东';

SELECT *from students where hometown in （ ‘北京’ , '上海', '广东'）;

SELECT *from students where age >= 25 and age <= 30;
SELECT *from students where age BETWEEN 25 and 30;
```

28.空判断

- null不是0，也不是''，null在SQL里面代表空，什么也没有
- null不能用比较运算符判断
- is null ---是否为null
- is not null ---是否不为null
    - 不能用字段名 = null 字段名 != null 这些都是错误的

```sql
SELECT *from students where card is null;

SELECT *from students where card is not null;
```

29.where己句可以用到update和delete语句后面

```sql
update students set class='2班' where age = 25 and name ='孙尚香';

DELETE from students where class = '1班' and age > 30;
```

30.order by 排序

- order by 字段名[asc/dsc]
    - asc代表从小到大，升序，asc可以省略
    - desc代表从大到小，不可以省略

```sql
-- 例1：查询所有学生记录，按age 年龄从小到大排序

select *from students order by age asc;
select *from students order by age;

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

- 排序的另一个例子

```sql
-- 例2：查询所有学生记录，按age年龄从大到小排序，
-- 年龄相同时，再按 studentsNo 学号从小到大排序
SELECT*from students ORDER BY age desc, studentsNo;
```

- 当一条select语句出现了where和order by

- select *from 表名 where 条件 order by 字段1，字段2；
- 一定要把where写在order前面；

```
SELECT *from students where sex = '男' order by class, studentsNO desc;
```

31.聚合函数

count求select返回的记录总数

- count（字段名）

```
-- 查询学生总数（查询students表有多少条记录）
select count(*) from students;
select count(name) from students;
select count(DISTINCT class) from students;
select count(DISTINCT sex) from students;

-- 查询女同学的数量
SELECT count (name) from students where sex = '女';
SELECT count(*) from students wheere sex = '女';
SELECT from(sex) from students where sex = '女';
```

max查询最大值

- max（字段名）
- 查询指定字段里的最大值

```
-- 查询students中的最大年龄
SELECT max(age) from students;

-- 查询students中的女生最大年龄
SELECT max(age) from students where sex = '女';

-- 查询students中的"1班"的最大年龄
SELECT max(age) from students where class = '1班';
```

- 聚合函数不能用到where后面的条件里

min查询最小值

- min（字段名）
- 查询指定字段里的最小值

```
-- 查询students中的最小年龄
SELECT min(age) from students;

-- 查询students中的女生最小年龄
SELECT min(age) from students where sex = '女';

-- 查询students中的"1班"的最小年龄
SELECT min(age) from students where class = '1班';
```

sum求和

- sum（字段名）
- 指定字段的值求和

```
-- 查询students中的年龄总和
SELECT sum(age) from students;

-- 查询students中的女生年龄总和
SELECT sum(age) from students where sex = '女';

-- 查询students中的"1班"的年龄总和
SELECT sum(age) from students where class = '1班';
```

avg求平均数

- avg（字段名）
- 指定字段的平均值

```
-- 查询students中的平均年龄
SELECT avg(age) from students;

-- 查询students中的女生平均年龄
SELECT avg(age) from students where sex = '女';

-- 查询students中的"1班"的平均年龄
SELECT avg(age) from students where class = '1班';
```

- avg字段中如果有null，null不做为分母计算平均

32.数据分组

- group by 字段名
- select 聚合函数 from 表名 where 条件 group by 字段
- select 聚合函数 from 表名 group by 字段
- group by就是配合聚合函数使用的

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

- group by的例子

```
-- 分别查询各个年龄的同学数量

select age, count(*) from students group by age;
```

- where与group by

```sql
-- 分别查询'1班'不同性别学生数量

select sex, count(*) from students where class = '1班' group by sex;
```

```sql
-- 练习：统计各个班级学生总数、平均年龄、最大年龄、最小年龄
-- 但不统计'3班'。统计结果按班级名称从大到小排序
SELECT class, count(age), avg(age), max(age), min(age) from students where class
<> '3班' GROUP BT class ORDER BY class desc;
```

- where和group by和order by的顺序
  - select *from 表名 where 条件 group by 字段 order by 字段

33.分组聚合之后的数据筛选

- having子句
- 总是出现在group by之后
- select *from 表名 group by 字段 having 条件

```sql
-- 用where查询男生总数
-- where先筛选复合条件的记录，然后在聚合统计
SELECT count(*) from students where sex = '男';

-- 用having查询男生总数
-- having先分组聚合统计，在统计的结果中筛选
SELECT count(*) from students GROUP BY sex HAVING sex = '男';
```

34.having配合聚合函数的使用

- where后面条件不能使用聚合函数，having可以使用聚合函数

```sql
-- 求班级人数大于3人的班级名字
select class from students GROUP BT class HAVING count(*)>3;
```

35.having与where筛选的区别

- where是对标的原始数据进行筛选
- having是对group by之后已经分组过的数据进行筛选
- having可以使用聚合函数，where不能使用聚合函数

36.limit显示指定的记录数

- select *from 表名 where 条件 group by 字段 order by 字段 limit, count
- limit总是出现在select语句的最后
- start代表开始行号，行号从0开始编号
- count代表要显示多少行
- 省略start默认从0开始，从第一行开始

```sql
-- 查询前三记录
SELECT *from students limit 0, 3;
SELECT *from students limit 3;

-- 查询从第4条记录开始的三条记录
SELECT *from students limit 3, 3;
```

37.数据分页显示

- m 每页显示多少条记录
- n,第n页
- (n-1)*m, m
- 把计算结果写到limit后面

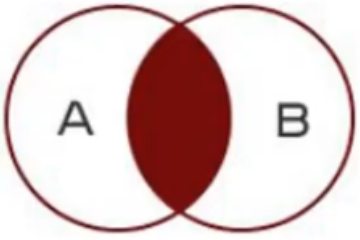
```sql
-- 每页显示4条记录，第3页的结果
select *from students limit 8, 4;

-- 每页显示4条，第2页的结果
select *from students limit 4, 4;
```

- 已知每页记录数，求一张表需要几页显示完
  - 求总页数
  - 总页数/每页的记录数
  - 如果结果是整数，那么就是总页数，如果结果有小数，那么就在结果的整数+1

38.连接查询

- 内连接
  - 把两张表相同的地方查询出来



- 左连接
  - 包括了内连接，同时还查询左表特有的内容

- 右连接

- 包括了内连接，同时还查询右表特有的内容

39.内连接

- 语法
  - select *from 表1 inner join 表2 on 表1.字段 = 表2;
  - 内连接最重要的是，找两张表要关联的字段

```
SELECT * from a INNER JOIN b on a.id = b.id;
```

- students表和scores内连接查询结果

```
SELECT * from students INNER JOIN scores on students.studentNo = scores.studentNo;
```

students表和scores表通过studentNO字段关联

studentNo	name	sex	hometown	age	class	card	id	courseNo	studentNo	score
1	王昭君	女	北京	30	1班	1.1E+17	1	1	1	90
2	诸葛亮	男	上海	29	2班	3.1E+17	2	1	2	75
2	诸葛亮	男	上海	29	2班	3.1E+17	3	2	2	98
1	王昭君	女	北京	30	1班	1.1E+17	4	3	1	86
3	张飞	男	南京	30	3班	3.2E+17	5	3	3	80
4	白起	男	安徽	35	4班	3.4E+17	6	4	4	79
5	大乔	女	天津	28	3班	1.2E+17	7	5	5	96
6	孙尚香	女	河北	25	1班	1.31E+17	8	6	6	80

- 隐式连接语法
  - 语法：select *from 表1，表2 where 两个表的连接条件

```
-- 隐式内连接
SELECT *from students, scores where students.studentNo = scores.studentNo;
```

- 内连接查询，显示指定的字段

```
-- students表与scores内连接，只显示name 课程号 成绩
SELECT name, courseNO, score from students INNER JOIN scores on
students.studentNO = scores.studentNo;
```

- 表的别名在查询中的使用

```
SELECT name 姓名, courseNo 课程编号, score 成绩 from students st INNER JOIN
scores sc on st.studentNo = sc.studentNo;
```

- 带有where的内连接
  - 语法：select*from 表1 inner join 表2 on 表1.字段=表2.字段 where 条件

```sql
-- 查询王昭君的信息，要求只显示姓名、课程号、成绩
select name, courseNo, score from students s1 INNER JOIN scores s2 on
s1.studentNo = s2.studentNo where s1.name = '王昭君';
```

- 带有and的where条件

```sql
-- 查询姓名为'王昭君'，并且成绩小于90的信息，要求只显示姓名、成绩
select name, score from students s1 INNER JOIN scores s2 on s1.studentNo =
s2.studentNo where s1.name = '王昭君' and s2.name < 90;
```

- 多表内连接

```sql
-- 例 7：查询学生信息和成绩以及成绩对应的课程名称
SELECT * from students inner join scores on students.studentNo = scores.studentNo
inner join courses on scores.courseNo = courses.courseNo;
```

sql

40.写SQL三步法

- 搭框架
  - 基本的select语句框架搭建起来，如果有多表，把相应的多表也联合起来
- 看条件按
  - 决定where后面的具体条件
- 显示的字段
  - select后面到底要显示什么字段

```sql
-- 练习 2：查询所有学生的'linux'课程成绩，要求只显示姓名、成绩、课程名
-- 第一步:搭框架
-- SELECT * from students INNER JOIN scores
-- on students.studentNo = scores.studentNo
-- INNER JOIN courses on scores.courseNo = courses.courseNo;
-- 第二步:看条件
-- SELECT * from students INNER JOIN scores
-- on students.studentNo = scores.studentNo
-- INNER JOIN courses on scores.courseNo = courses.courseNo
-- where courseName = 'linux';
-- 第三步:返回字段名
SELECT name, score, courseName from students INNER JOIN scores
on students.studentNo = scores.studentNo
INNER JOIN courses on scores.courseNo = courses.courseNo
where courseName = 'linux';
```

41.左连接

- 语法
  - select*from 表1 left join 表2 on 表1.字段 = 表2.字段

```sql
-- 查询所有学生的信息以及成绩，包括没有成绩的学生
SELECT *from students left JOIN scores ON students.studentNo =
scores.studentNo;
```

42.右连接

- 语法
  - select *from 表1 right join on 表1.字段 = 表2.字段

```sql
-- 查询所有课程的信息， 包括没有成绩的课程
SELECT *from scores RIGHT JOIN courses ON scores.courseNo = courses.courseNo;
```

43.多表联合查询，同名字段的处理方式

- 如果一条select要用到多个表，表中有同名字段，就需要表名，字段名加以区分

```sql
select students.studentNo from students INNER JOIN scores ON
students.studentNo = scores.studentNo;
```

44.自关联

```sql
-- 查询一共有多少个省
SELECT count(*) from areas where pid is null;
-- 查询有多少市
SELECT count(*) from areas where pid is not null;
```

- 自关联，是同一张表做连接查询
- 自关联下，一定找到同一张可关联的不同字段

```sql
-- 查询广东省的所有城市
SELECT *from areas a1 INNER JOIN area a2 on a1.id = a2.pid WHERE a1.name = '广
东省';
```

45.子查询

- 子查询是嵌套到主查询里面的
- 子查询作为主查询的数据源或者条件
- 子查询是独立可以单独运行的查询语句
- 主查询不能独立运行，依赖子查询的结果

```sql
SELECT avg(age) from students;
SELECT *from students where age>30.1667;

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

- 标量子查询------子查询返回结果只有一行，一列

```
-- 例 2：查询 30 岁的学生的成绩
-- 1,查询30岁学生的studentNO
-- select studentNo from students where age = 30;
--
-- SELECT * from scores where studentNo in ('001', '003', '011');

-- 用子查询实现
SELECT * from scores where studentNo in
(select studentNo from students where age = 30);

```

- 列子查询----------子查询返回一列多行
- 表级子查询----------子查询返回结果为多行，多列

```
-- 例 3：用子查询，查询所有女生的信息和成绩
-- 用内连接实现
SELECT * from students INNER JOIN scores ON
students.studentNo = scores.studentNo
where sex = '女';
-- 用子查询实现
select * from (SELECT * from students where sex = '女') stu
INNER JOIN scores sc on stu.studentNo = sc.studentNo;
```

46.contact拼接字符串函数

- contact(参数1，参数2，参数3，参数n)
  - 参数可以是数字，也可以是字符串
  - 把所有的参数连接成一个完整的字符串

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

47.length返回字符串字符的个数

- 一个utf8格式的汉字，length也返回3

```sql
-- 例 2：计算字符串'abc'的长度
select length('abc');

-- 例 3：计算字符串'我和你'的长度
SELECT length('我和你');


-- 例 4：计算字符串'我和you'的长度
SELECT length('我和you');

```

48.mysql内置函数可以在where条件后使用

```sql
SELECT *from students where length(name) = 9;
```

49.left从字符串左侧截取指定数量字符

- left(字符串, n)
  - n代表从字符串左侧截取n个字符

```sql
-- 例 5：截取字符串'我和你abc'的左端 3 个字符
select left('我和你abc', 3);

-- 例 6：截取字符串'我和你abc'的左端 4 个字符
select left('我和你abc', 4);

-- 例 6：截取字符串'abc我和你'的左端 4 个字符
select left('abc我和你', 4);
```

50.right从字符串右侧截取指定数量字符

- right(字符串, n)
  - n代表从字符串右侧截取n个字符

```sql
-- 例 5：截取字符串'我和你abc'的右端 3 个字符
select right('我和你abc', 3);

-- 例 5：截取字符串'我和你abc'的右端 4 个字符
select right('我和你abc', 4);
```

51.substring从字符串指定位置截取指定数量字符

- substring(字符串, 起始位置, n)
  - 起始位置从1开始
  - n代表截取的数量

```
-- 例 7：截取字符串'我和你abc'从第 2 个字符开始的 3 个字符
select substring('我和你abc', 2, 3);


-- 例 7：截取字符串'我和你abc'从左侧开始的 3 个字符
select substring('我和你abc', 1, 3);

-- 例 7：截取字符串'我和你abc'从第 4 个字符开始的 1 个字符
select substring('我和你abc', 4, 1);
```

52.内置函数可以用在select显示的字段名中

```
-- 例 8：截取 students 表中所有学生的姓
SELECT left(name, 1) from students;
SELECT substring(name, 1, 1) from students;
```

53.ltrim去除字符串左侧空格

- ltrim(带空格的字符串)

```
-- 例1：去除字符串'     abcd'左侧空格

SELECT ltrim('     abcd');
```

54.rtrim去除字符串左侧空格

- rtrim(带空格的字符串)

```
-- 例1：去除字符串'   abcd     '右侧空格
SELECT ltrim('   abcd     ');
SELECT contact(ltrim('   abcd     '), '测试字符');
```

55.trim去除字符串两个空格

- trim(带空格的字符串)

```
-- 例1：去除字符串'   abcd   '两侧空格
SELECT trim('   abcd   ');
```

56.round四舍五入

- round(数字, d)
  - d代表要保留的小数位，省略d默认为0

```
-- 例 1: 1.653 四舍五入，保留整数位
SELECT round(1.653);


-- 例 2: 1.653 四舍五入，保留2位小数
SELECT round(1.653, 2);

```

57.rand随机数

- rand()
  - 每次运行会产生一个从0到1之间的浮点数
- 经常用rand对一张表进行随机排序
  - order by rand()

```
select rand();

-- 小技巧：从学生表中随机抽出一个学生
SELECT *from students order by rand() LIMIT 1;
```

58.current_date返回系统日期

- current_date()

59.current_time返回系统时间

- current_time()

60.current_date返回系统日期

- now()

```sql
select curren_date();

select current_time();

select cuurent_date();
```

61.存储过程

```sql
-- 例 1：创建存储过程 stu()，查询 students 表所有学生信息
CREATE PROCEDURE stu()
BEGIN
    SELECT * from students;
end

-- 调用存储过程stu
call stu();

-- 删除存储过程，删除的时候不用写名字后面的()
DROP PROCEDURE stu;
drop PROCEDURE if EXISTS stu;
```

62.视图

- 视图就是对select语句的封装
- 视图可以理解为一张只读的表，针对视图只能用select，不能用delete和update

```sql
-- 创建一个视图，查询所有男生信息

CREATE VIEW stu_nan as
SELECT * from students where sex = '男';

-- 使用视图
SELECT * from stu_nan INNER JOIN scores
on stu_nan.studentNo = scores.studentNo;

-- 删除视图
drop VIEW stu_nan;
DROP view if EXISTS stu_nan;
```

63.事务

- 事务是多条更改数据的sql语句集合
- 一个集合数据有一致性，要么就都失败，要么就都成功
- begin ---开始事务
- rollback ---回归事务，放弃对表的修改
- commit ---提交事务，对表的修改生效

没有写begin代表没有事务，没有事务的表操作都是实时生效

如果只写了begin，没有rollback，也没有commit，系统推出，结果是rollback

回滚事务的操作

```
-- 例 1：开启事务，
-- 删除 students 表中 studentNo 为 001 的记录，
-- 同时删除 scores 表中 studentNo 为 001 的记录，
-- 回滚事务，两个表的删除同时放弃
-- 开始事务
begin;
DELETE from students where studentNo = '001';
DELETE from scores where studentNo = '001';
-- 回滚事务,放弃更改
ROLLBACK;


SELECT * from students;
SELECT * from scores;
```

如果开始了一个事务，执行了begin，之后，没有rollback，也没有commit，中间系统出问题了，默认会执行rollback

提交事务

```
-- 例 2：开启事务，
-- 删除 students 表中 studentNo 为 001 的记录，
-- 同时删除 scores 表中 studentNo 为 001 的记录，
-- 提交事务，使两个表的删除同时生效

begin;
DELETE from students where studentNo = '001';
DELETE from scores where studentNo = '001';
-- 提交事务，一旦提交事务，两个删除操作同时生效
commit;
```

64.索引

- index
- 给表建立索引，目的是加快select查询的速度
- 如果一个表记录很少，几十条，或者几百条，不要索引
- 表的记录特别多，如果没有索引，select语句效率非常低

创建索引

- create index 索引名 on 表名(字段)
- 如果字段为字符串，需要写明创建表字段的时候字符串的长度

```
-- 例 1：为表 students 的 age 字段创建索引，名为 age_index
CREATE index age_index on students (age);

-- 例 2：为表 students 的 name 字段创建索引，名为 name_index
CREATE INDEX name_index on students (name(10));
```

调用索引

- 不需要显示的写调用索引的语句，只要where条件后面用到的字段建立了索引，那么系统会字段调用

```sql
-- where条件后面的字段,数据库系统会自动查找是否有索引
-- 这里会自动调用age_index
select * from students where age = 30;
-- 自动调用name_index
SELECT * from students where name = '李白';
-- 不会调用任何索引,因为sex字段没有索引
SELECT * from students where sex = '女';
```

查看索引

- show index from 表名
- 对于主键，系统会自动建立索引

```sql
-- 查看students的索引
show index from students;
```

删除索引

- drop index 索引名 on 表名

```sql
show index from students;

-- 删除索引age_index
drop index age_index on students;

-- 删除索引name_index
deop index name_index on students;
```

索引的优缺点

- 提高select的查询速度
- 降低update，delete和insert语句的执行速度
- 项目中百分之80%以上是select，所以select是必须的
- 在实际工作如果涉及大量的数据修改操作，修改之前可以把索引删除，修改完成后再把索引建立起来

65.基于命令行的mysql

- mysql -h mysql服务器的地址 -u 用户名 -p
  - -h 如果是使用本机的mysql，-h可以省略

mysql登录之后的常用命令

- show databases
  - 显示系统所有的数据库
- use 数据库名
  - 使用指定的一个数据库

```
-- 使用mydb数据库
use mydb
```

- show tables
  - 查看指定数据库有多少表
- 如果命令行默认字符集与数据库默认字符集不同
  - 在windows默认字符集是gbk
  - set names gbk
    - 告诉mysql，客户端的字符集是gbk
- 在命令行中每条sql语句用;结尾
- 可以通过desc 表名 查看一个表的字段结构
  - desc students
  - 查看students每个字段的定义
- 在命令行下创建和删除数据库
  - create database 数据库名 default charset 字符集

```
-- 创建一个数据库mytest，默认字符集utf8
create database mytest dafault charset utf8;

drop database mytest;
drop database if exists mytest;
```