第05天 MySQL数据库

**【学习目标】理解、了解、应用、记忆**

1. **【理解】MySQL单表查询**
2. 【应用】独立编写SQL语句按照指定字段进行排序
3. 【应用】独立编写SQL语句使用聚合函数完成相关数据统计
4. 【应用】独立编写SQL语句使用分组完成相关数据统计
5. 【应用】能够按照文档完成备份数据库和还原数据库操作
6. **【应用】SQL约束**
7. 【应用】独立编写SQL语句完成主键约束的添加和删除
8. 【应用】独自编写SQL定义自动增长列
9. 【应用】独立编写SQL定义非空约束的字段
10. 【应用】独立编写SQL给表中字段设置唯一约束
11. **【应用】MySQL多表操作**
12. 【理解】阐述表与表之间的关系分别有哪些
13. 【应用】独立编写SQL使用外键关键关联两张表
14. 【应用】独立编写SQL实现一对多的表关系并完成相关操作
15. 【应用】独立编写SQL实现多对多的表关系并完成相关操作

# MySQL单表查询

## SQL单表查询--排序

### 排序格式

通过order by语句，可以将查询出的结果进行排序。放置在select语句的最后。

格式:

SELECT \* FROM 表名 where 条件 ORDER BY 排序字段 ASC|DESC;

ASC 升序 (默认)

DESC 降序

#1.使用价格排序(降序)

SELECT \* FROM d\_product ORDER BY price DESC;

#2.在价格排序(降序)的基础上，以分类排序(降序)

SELECT \* FROM d\_product ORDER BY price DESC,category\_id DESC;

#3.显示商品的价格(去重复)，并排序(降序)

SELECT DISTINCT price FROM d\_product ORDER BY price DESC;

### 案例练习一

/\*排序\*/

/\*1、默认按照主键升序排列\*/

/\*select \* from d\_product;\*/

/\*2、采用order by关键字排列\*/

/\*按照价格升序排列\*/

/\*select \* from d\_product order by price;/\*默认升序 asc\*/

/\*按照价格降序排列\*/

/\*select \* from d\_product order by price desc;\*/

/\*2、两阶排序\*/

/\*在价格排序(降序)的基础上，以分类排序(降序)\*/

/\*select \* from d\_product order by price desc,category\_id desc;\*/

/\*结合列筛选和行筛选\*/

select pname,price from d\_product where pname like'%o%' order by price desc;

## SQL单表查询--聚合

之前我们做的查询都是横向查询，它们都是根据条件一行一行的进行判断，而使用聚合函数查询是纵向查询，它是对一列的值进行计算，然后返回一个单一的值；另外聚合函数会忽略空值。

今天我们学习如下五个聚合函数：

* count：统计指定列不为NULL的记录行数；
* sum：计算指定列的数值和，如果指定列类型不是数值类型，那么计算结果为0；
* max：计算指定列的最大值，如果指定列是字符串类型，那么使用字符串排序运算；
* min：计算指定列的最小值，如果指定列是字符串类型，那么使用字符串排序运算；
* avg：计算指定列的平均值，如果指定列类型不是数值类型，那么计算结果为0；

### 案例练习二

/\*聚合函数\*/

/\*1、count(列名) 统计个数\*/

/\*统计价格在800以上一共有多少商品\*/

/\*SELECT COUNT(\*) as total from d\_product where price>=800;\*/

/\*统计一共有几种价格\*/

/\*select count(DISTINCT price)from d\_product;\*/

/\*2、sum 计算和值\*/

/\*统计价格的和值\*/

/\*select sum(price) from d\_product;\*/

/\*3、max 获取最大值\*/

/\*获取价格最大的商品的价格\*/

/\*select max(price) from d\_product;\*/

/\*4、min 获取最小值\*/

/\*获取价格最小的商品的价格\*/

/\*select min(price) from d\_product;\*/

/\*5、avg 获取平均值\*/

/\*获取所有商品的平均价格\*/

select avg(price) from d\_product;

## SQL单表查询--分组

分组查询是指使用group by字句对查询信息进行分组。

* 格式：

SELECT 字段1,字段2… FROM 表名 GROUP BY分组字段 HAVING 分组条件;

分组操作中的having子语句，是用于在分组后对数据进行过滤的，作用类似于where条件。

* having与where的区别:
  + having是在分组后对数据进行过滤.

where是在分组前对数据进行过滤

* + having后面可以使用分组函数(统计函数)

where后面不可以使用分组函数。

### 案例练习三

/\*分组 最好和聚合函数结合使用\*/

/\*按照价格分组\*/

/\*select price from d\_product group by price;\*/

/\*按照类别id分组 该类名的商品个数 平均价格 类别\*/

/\*select count(\*),avg(price),category\_id from d\_product group by category\_id;\*/

/\*行筛选之后再分组\*/

/\*先行筛选再分组：对非空的category\_id进行分组 注意where的位置\*/

/\*select count(\*),avg(price),category\_id from d\_product where category\_id is not null group by category\_id;\*/

/\*先分组再筛选\*/

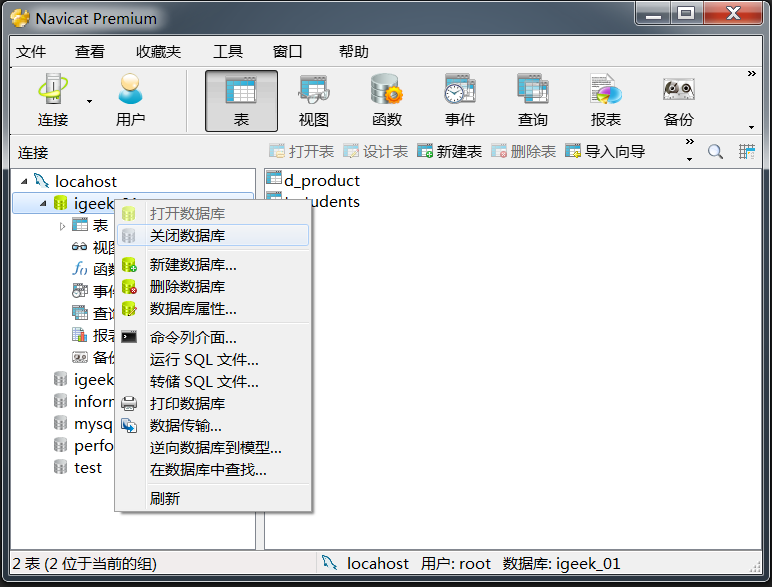
/\*筛选出评价价格在500以上的商品分组\*/

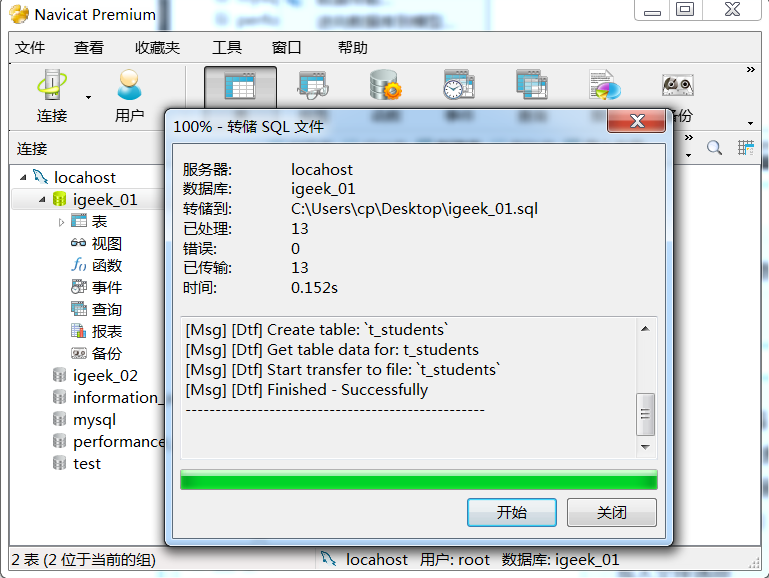
select count(\*),avg(price),category\_id from d\_product group by category\_id having avg(price)>=500;

## 数据库的备份与还原

### SQL备份

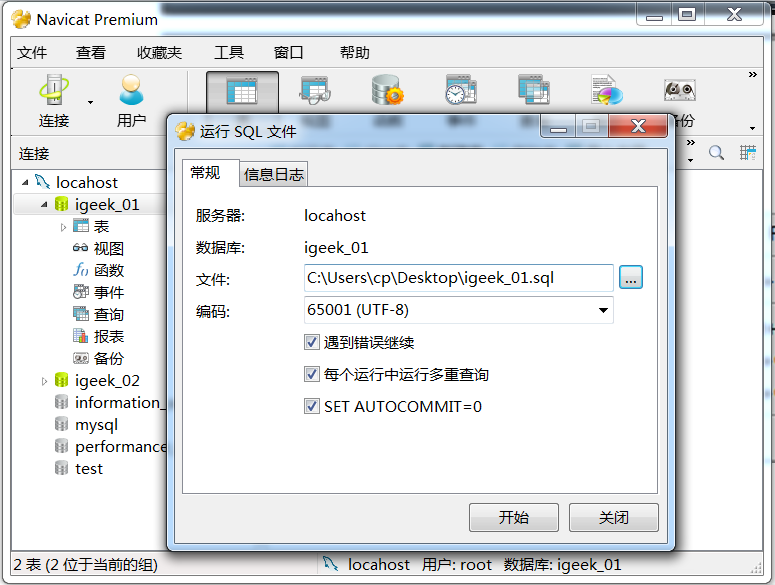
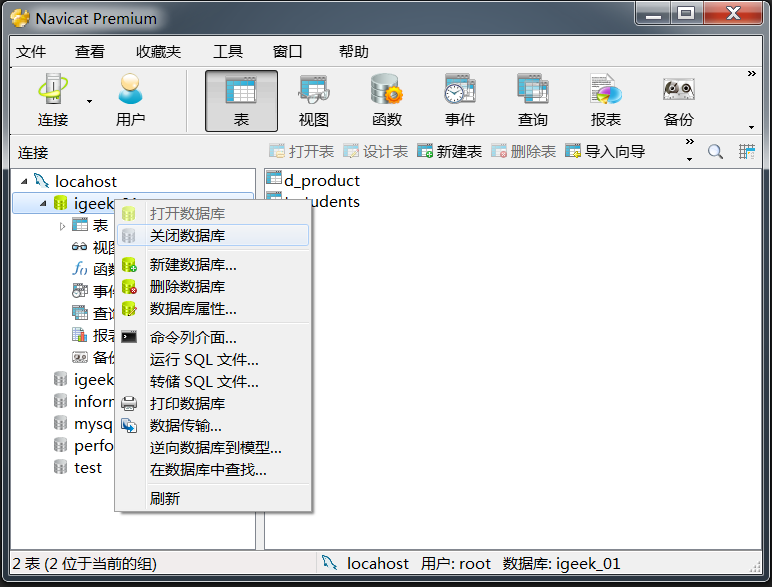
选中数据库，右键 ”转储SQL文件” ， 指定导出路径，保存成.sql文件即可。

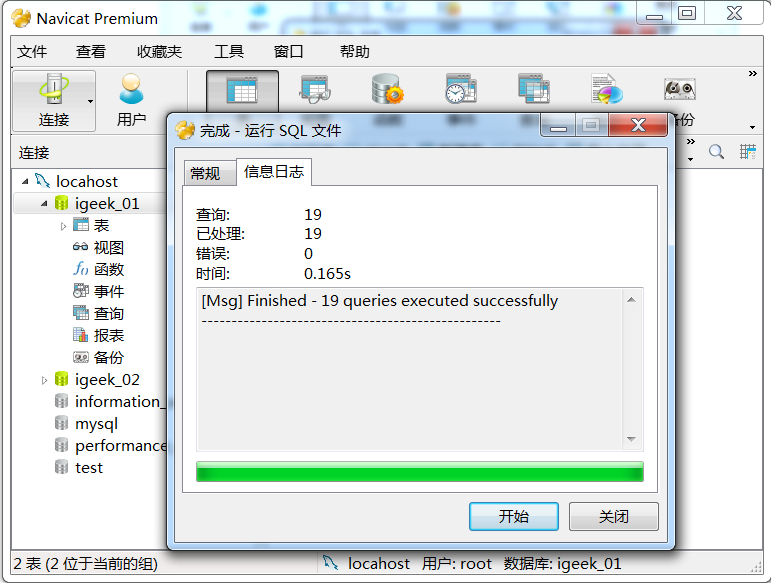




### SQL恢复

数据库列表区域右键“运行SQL文件”， 指定要执行的SQL文件，点击“开始”即可。





# SQL约束

## SQL约束--主键约束

PRIMARY KEY 约束唯一标识数据库表中的每条记录。

主键必须包含唯一的值。

主键列不能包含 NULL 值。

每个表都应该有一个主键，并且每个表只能有一个主键。

### 添加主键约束

* 方式一：创建表时，在字段描述处，声明指定字段为主键：

CREATE TABLE Persons

(

Id\_P int PRIMARY KEY,

LastName varchar(255),

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

* 方式二：创建表时，在constraint约束区域，声明指定字段为主键：
  + 格式：[constraint 名称] primary key (字段列表)
  + 关键字constraint可以省略，如果需要为主键命名，constraint不能省略，主键名称一般没用。
  + 字段列表需要使用小括号括住，如果有多字段需要使用逗号分隔。声明两个以上字段为主键，我们称为联合主键。

CREATE TABLE Persons

(

FirstName varchar(255),

LastName varchar(255),

Address varchar(255),

City varchar(255),

CONSTRAINT pk\_PersonID PRIMARY KEY (FirstName,LastName)

)

或

CREATE TABLE Persons

(

FirstName varchar(255),

LastName varchar(255),

Address varchar(255),

City varchar(255),

PRIMARY KEY (FirstName,LastName)

)

* 方式三：创建表之后，通过修改表结构，声明指定字段为主键：

ALTER TABLE Persons ADD [CONSTRAINT 名称] PRIMARY KEY (字段列表)

CREATE TABLE Persons

(

FirstName varchar(255),

LastName varchar(255),

Address varchar(255),

City varchar(255)

)

ALTER TABLE Persons ADD PRIMARY KEY (FirstName,LastName)

### 删除主键约束

如需撤销 PRIMARY KEY 约束，请使用下面的 SQL：

ALTER TABLE Persons DROP PRIMARY KEY

## SQL约束--自动增长列

### 自动增长

我们通常希望在每次插入新记录时，数据库自动生成字段的值。

我们可以在表中使用 auto-increment（自动增长列）关键字，自动增长列类型必须是整形，自动增长列必须为键(一般是主键)。

* 下列 SQL 语句把 "Persons" 表中的 "P\_Id" 列定义为 auto-increment 主键

CREATE TABLE Persons

(

P\_Id int PRIMARY KEY AUTO\_INCREMENT,

LastName varchar(255),

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

* 向persons添加数据时，可以不为P\_Id字段设置值，也可以设置成null，数据库将自动维护主键值：

INSERT INTO Persons (FirstName,LastName) VALUES ('Bill','Gates')

INSERT INTO Persons (P\_Id,FirstName,LastName) VALUES (NULL,'Bill','Gates')

* 面试：delete和truncate的区别
  + Delete删除表中的数据，但不重置auto-increment记录数。
  + Truncate删除表中的数据，auto-increment记录数将重置。Truncate其实先删除表然后再创建表。
* 扩展：默认地，AUTO\_INCREMENT 的开始值是 1，如果希望修改起始值，请使用下列 SQL 语法：

ALTER TABLE Persons AUTO\_INCREMENT=100

### 案例练习四

##自动增长列

CREATE TABLE ai01(

id INT PRIMARY KEY AUTO\_INCREMENT,

NAME VARCHAR(50)

);

# 向表中插入数据

INSERT INTO ai01(NAME) VALUES('a');

INSERT INTO ai01(id,NAME) VALUES(NULL,'a');

## 自动增长列非法使用

CREATE TABLE ai02(

id VARCHAR(32) PRIMARY KEY AUTO\_INCREMENT, #字段描述错误，类型不对

NAME VARCHAR(50)

);

CREATE TABLE ai03(

id INT AUTO\_INCREMENT, #自动增长列必须是键（一般是主键）

NAME VARCHAR(50)

);

#扩展：设置自动增长列初始值，id默认值为1

CREATE TABLE ai04(

id INT PRIMARY KEY AUTO\_INCREMENT,

NAME VARCHAR(50)

);

ALTER TABLE ai04 AUTO\_INCREMENT = 100;

INSERT INTO ai04(NAME) VALUES('a');

#面试题 ：delete 和 truncate 区别

## delete 删除表数据，但不重置自动增长列记录数。

##truncate 删除表数据，重置自动增长列的记录数。（先删除表，再创建表）

DELETE FROM ai01;

TRUNCATE TABLE ai01;

## SQL约束--非空约束

NOT NULL 约束强制列不接受 NULL 值。

NOT NULL 约束强制字段始终包含值。这意味着，如果不向字段添加值，就无法插入新记录或者更新记录。

* 下面的 SQL 语句强制 "Id\_P" 列和 "LastName" 列不接受 NULL 值：

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

## SQL约束--唯一约束

UNIQUE 约束唯一标识数据库表中的每条记录。

UNIQUE 和 PRIMARY KEY 约束均为列或列集合提供了唯一性的保证。

PRIMARY KEY 拥有自动定义的 UNIQUE 约束。

请注意，每个表可以有多个 UNIQUE 约束，但是每个表只能有一个 PRIMARY KEY 约束。

### 添加唯一约束

与主键添加方式相同，共有3种，

* 方式1：创建表时，在字段描述处，声明唯一：

CREATE TABLE Persons

(

Id\_P int UNIQUE,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

* 方式2：创建表时，在约束区域，声明唯一：

CREATE TABLE Persons

(

Id\_P int,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

CONSTRAINT 名称UNIQUE (Id\_P)

)

* 方式3：创建表后，修改表结构，声明字段唯一：

ALTER TABLE Persons ADD [CONSTRAINT 名称] UNIQUE (Id\_P)

### 删除唯一约束

* 如需撤销 UNIQUE 约束，请使用下面的 SQL：

ALTER TABLE Persons DROP INDEX 名称

如果添加唯一约束时，没有设置约束名称，默认是当前字段的字段名。

### 案例练习五

##唯一约束

CREATE TABLE un01(

id VARCHAR(32),

NAME VARCHAR(50)

);

ALTER TABLE un01 ADD CONSTRAINT u0001 UNIQUE (id);

#删除

ALTER TABLE un01 DROP INDEX u0001;

##如果没有设置约束名称，唯一约束名称默认是字段名称

CREATE TABLE un02(

id VARCHAR(32),

NAME VARCHAR(50)

);

ALTER TABLE un02 ADD UNIQUE (id);

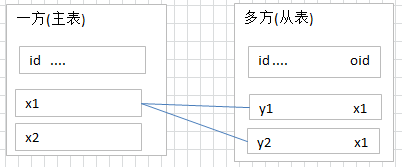
#删除

ALTER TABLE un02 DROP INDEX id;

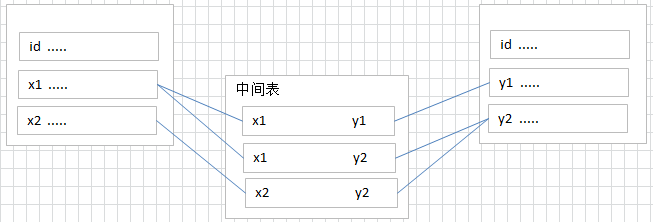
# 多表操作

## 表与表之间的关系

* 一对多关系：
  + 常见实例：客户和订单，分类和商品，部门和员工.
  + 一对多建表原则：在从表(多方)创建一个字段，字段作为外键指向主表(一方)的主键.



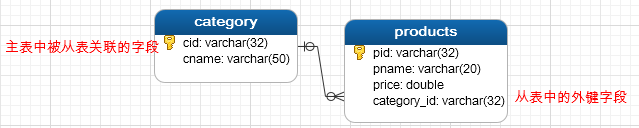
* 多对多关系：
  + 常见实例：学生和课程、用户和角色
  + 多对多关系建表原则：需要创建第三张表,中间表中至少两个字段，这两个字段分别作为外键指向各自一方的主键.



* 一对一关系：(了解)
  + 在实际的开发中应用不多.因为一对一可以创建成一张表.
  + 两种建表原则：
    - 外键唯一：主表的主键和从表的外键（唯一），形成主外键关系，外键唯一unique。
    - 外键是主键：主表的主键和从表的主键，形成主外键关系。

## 外键约束

现在我们有两张表“分类表”和“商品表”，为了表明商品属于哪个分类，通常情况下，我们将在商品表上添加一列，用于存放分类cid的信息，此列称为：外键



此时“分类表category”称为：主表，“cid”我们称为主键。“商品表products”称为：从表category\_id称为外键。我们通过主表的主键和从表的外键来描述主外键关系，呈现就是一对多关系。

外键特点：

* + - 从表外键的值是对主表主键的引用。
    - 从表外键类型，必须与主表主键类型一致。
* 声明外键约束

语法：alter table 从表 add [constraint] [外键名称] foreign key (从表外键字段名) references 主表 (主表的主键);

[外键名称] 用于删除外键约束的，一般建议“\_fk”结尾

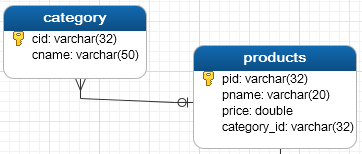
alter table 从表 drop foreign key 外键名称

* 使用外键目的：
  + 保证数据完整性

## 一对多关系

### 一对多的表关系的分析与实现

#### 分析



* category分类表，为一方，也就是主表，必须提供主键cid
* products商品表，为多方，也就是从表，必须提供外键category\_id

#### 实现：分类和商品

### 案例练习六

/\*使用数据库\*/

USE igeek\_02;

####一对多关系

#1 创建主表：分类表

CREATE TABLE category(

cid VARCHAR(32) PRIMARY KEY,

cname VARCHAR(50)

);

#2 创建从表：商品表

CREATE TABLE products(

pid VARCHAR(32) PRIMARY KEY,

pname VARCHAR(50),

price DOUBLE,

category\_id VARCHAR(32)

);

#3 将使用 主外键关系进行描述

ALTER TABLE products ADD CONSTRAINT product\_fk FOREIGN KEY (category\_id) REFERENCES category (cid);

### 一对多关系操作

### 案例练习七

use igeek\_02;

/\*一对多数据操作\*/

#1、向商品分类表插入数据

INSERT INTO category(cid,cname) VALUES('c001','电脑');

#2、向商品表插入数据,不含外键信息

INSERT INTO products(pid,pname,price) VALUES('p001','iphone X',8888);

#结论：外键可以不填写，默认就是null

#3、向商品表插入数据，包含外键信息(最常用)

INSERT INTO products(pid,pname,price,category\_id) VALUES('p002','联想笔记本',3000,'c001');

#4、向商品表插入数据,包含的外键信息在分类表中找不到

INSERT INTO products(pid,pname,price,category\_id) VALUES('p003','卫龙辣条',20,'c888');

#结论：从表中不能添加（或者更新）主表中不存在的外键数据

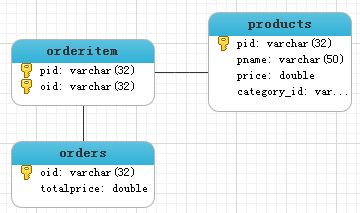
#5、删除主表的数据

DELETE FROM category WHERE cid='c001';

#结论：主表不能删除(或者更新)从表中已经使用到的数据

## 多对多关系

### 多对多表关系的分析和实现



* 商品和订单多对多关系，将拆分成两个一对多。
* products商品表，为其中一个一对多的主表，需要提供主键pid
* orders 订单表，为另一个一对多的主表，需要提供主键oid
* orderitem中间表，为另外添加的第三张表，需要提供两个外键oid和pid

### 案例练习八

use igeek\_02;

/\*多对多建表\*/

#1、商品表（已存在）

#2、订单表

CREATE TABLE orders(

oid VARCHAR(32) PRIMARY KEY,

totalprice double

);

#3、中间表

CREATE TABLE orderitem(

pid VARCHAR(32),

oid VARCHAR(32)

);

#4、两个外键

#4.1、中间表与商品表主外键

ALTER TABLE orderitem ADD CONSTRAINT orderitem\_products\_fk FOREIGN KEY(pid) REFERENCES products(pid);

#4.2、中间表与订单表主外键

ALTER TABLE orderitem ADD CONSTRAINT orderitem\_orders\_fk FOREIGN KEY(oid) REFERENCES orders(oid);

#5、扩展：中间表两个外键，一般会形成联合主键（2个组合在一起唯一）[可选]

ALTER TABLE orderitem ADD CONSTRAINT PRIMARY KEY(pid,oid);

### 多对多表关系的操作

### 案例练习九

use igeek\_02;

/\*多对多数据操作\*/

#1、向商品表插入数据

INSERT INTO products(pid,pname,price) VALUES('p003','IBM台式机',10000);

#2、向订单表插入数据

INSERT INTO orders(oid,totalprice) VALUES('x001',10000);

#3、向中间表插入数据（存在）

INSERT INTO orderitem(pid,oid) VALUES('p003','x001');

INSERT INTO orderitem(pid,oid) VALUES('p001','x001');

#4、删除中间表

DELETE FROM orderitem WHERE pid='p001' AND oid='x001';

#5、向中间表插入数据（不存在）

INSERT INTO orderitem(pid,oid) VALUES('p003','x002');

#注意：维护中间表，数据必须存在

#6、删除订单表（主表）数据（被使用中）

DELETE FROM orders where oid='x001';

#注意：主表中不能删除中间表正在使用的数据

#一对多与多对多的对比

#一对多在维护关系时，操作的是从表的外键（一个字段）

#多对多在维护关系时候，操作的时中间表的数据