day02-基础

学习目标：

- [ ] 查询
  - [ ] 能够使用SQL语句查询数据
  - [ ] 能够使用SQL语句进行条件查询
  - [ ] 能够使用SQL语句进行排序
  - [ ] 能够使用聚合函数
  - [ ] 能够使用SQL语句进行分组查询
- [ ] 能够完成数据的备份和恢复
- [ ] 能够使用可视化工具连接数据库，操作数据库
- [ ] 能够说出多表之间的关系及其建表原则
- [ ] 能够理解外键约束

第一章 SQL语句之DQL【重要、重要、重要】

语法：查询不会对数据库中的数据进行修改，根据指定的方式来呈现数据。

语法格式：

```
select * | 列名,列名  from 表名 [where 条件表达式]
```

- select 是查询指令，可以读 1 ~ n 行数据；
- 列名换成 * 号，可以查询所有字段数据；
- 使用 where 来指定对应的条件

1.1 准备工作

创建商品表

```
CREATE TABLE products (
    -- 自增加 AUTO_INCREMENT
    pid INT PRIMARY KEY AUTO_INCREMENT,
    pname VARCHAR(20),  -- 商品名称
    price DOUBLE,       -- 商品价格
    pdate DATE,         -- 日期
    sid VARCHAR(20)     -- 分类ID
);

INSERT INTO products VALUES(NULL,'泰国大榴莲', 98, NULL, 's001');
INSERT INTO products VALUES(NULL,'新疆大枣', 38, NULL, 's002');
INSERT INTO products VALUES(NULL,'新疆切糕', 68, NULL, 's001');
INSERT INTO products VALUES(NULL,'十三香', 10, NULL, 's002');
INSERT INTO products VALUES(NULL,'老干妈', 20, NULL, 's002');
```

1.2 简单查询

```sql
-- 查询所有的商品
select * from product;

-- 查询指定列：商品名和商品价格
select pname,price from product;

-- 别名查询，使用的 as 关键字，as 也可以省略的
-- 使用别名的好处：显示的时候使用识别性更强的名字，本身也不会去影响到表结构
-- 表别名
-- select 字段名 as 字段别名 from 表名
select * from product as p;

-- 列别名
-- select 列名 as 列别名 from 表名
select pname as pn from product;

-- 列和表，同时指定别名
-- select 列名 as 列别名 from 表名 as 表别名

-- 去掉重复值
-- select distinct 字段名 from 表名
select distinct pname from product;

--查询结果是表达式（运算查询）：将所有商品的价格 +10 元进行显示
-- select 列名+固定值 from 表名
-- select 列名1 + 列名2 from 表名
select pname, price + 10 from product;
```

1.3 条件查询

使用条件查询，可以根据当下具体情况直查想要的那部分数据，对记录进行过滤。

SQL 语法关键字： WHERE

语法格式：

```sql
select 字段名 from 表名 where 条件;
```

1. 运算符

操作符	描述	实例
=	等号，检测两个值是否相等，如果相等返回true	(A = B) 返回false。
<>, !=	不等于，检测两个值是否相等，如果不相等返回true	(A != B) 返回 true。
>	大于号，检测左边的值是否大于右边的值, 如果左边的值大于右边的值返回true	(A > B) 返回false。
<	小于号，检测左边的值是否小于右边的值, 如果左边的值小于右边的值返回true	(A < B) 返回 true。
>=	大于等于号，检测左边的值是否大于或等于右边的值, 如果左边的值大于或等于右边的值返回true	(A >= B) 返回false。
<=	小于等于号，检测左边的值是否小于于或等于右边的值, 如果左边的值小于或等于右边的值返回true	(A <= B) 返回 true。

```sql
-- 查询商品名称为十三香的商品所有信息
select * from product where pname = '十三香';

-- 查询商品价格 >60 元的所有的商品信息
select * from product where price > 60;
select * from product where price <= 60;

-- 不等于
select * from product where price != 60;
select * from product where price <> 60;
```

2. 逻辑运算符

NOT	逻辑非【!】
AND	逻辑与【&&】
OR	逻辑或【||】

```sql
select * from product where price > 40 and pid > 3;

select * from product where price > 40 or pid > 3;
```

3. in 关键字

```sql
-- in 匹配某些值中
select * from product where pid in (2,5,8);
-- 不在这些值中
select * from product where pid not in (2,5,8);
```

4. 指定范围中 between...and

```
select * from product where pid between 2 and 10;
```

5. 模糊查询 like 关键字

```
-- 使用 like 实现模糊查询
-- “新”开头
select * from product where pname like '新%';
-- 包含“新”
select * from product where pname like '%新%';
```

1.4 排序

语法:

```
select 字段名 from 表名 where 字段 = 值
order by 字段名 [asc | desc]

asc 升序
desc 降序
```

```
-- 查询所有的商品，按价格进行排序
select * from product order by price;

-- 查询名称有新的商品的信息并且按价格降序排序
select * from product where pname like '%新%' order by price desc;
```

1.5 聚合函数（组函数）

```
特点：只对单列进行操作

常用的聚合函数：
sum()：求某一列的和
avg()：求某一列的平均值
max()：求某一列的最大值
min()：求某一列的最小值
count()：求某一列的元素个数

-- 获得所有商品的价格的总和：
select sum(price) from product;

-- 获得所有商品的平均价格：
```

```sql
select avg(price) from product;

-- 获得所有商品的个数：
select count(*) from product;
```

1.6 分组查询

语法格式：

```sql
SQL 语法关键字：GROUP BY、HAVING

select 字段1, 字段2... from 表名
group by 分组字段
[having 条件];
```

```sql
-- 根据 cno 字段分组，分组后统计商品的个数
select sid, count(*) from product group by sid;

-- 根据 cno 分组，分组统计每组商品的平均价格，并且平均价格 > 60;
select sid, avg(price) from product group by sid having avg(price) > 60;
```

注意事项：

① select 语句中的列（非聚合函数列），必须出现在 group by 子句中

② group by 子句中的列，不一定要出现在 select 语句中

③ 聚合函数只能出现 select 语句中或者 having 语句中，一定不能出现在 where 语句中。

having 和 where 的区别：

```
首先，执行的顺序是有先有后。
```

1）where

```
对查询结果进行分组前，将不符合 where 条件的记录过滤掉，然后再分组。
where 后面，不能再使用聚合函数。
```

2）having

```
筛选满足条件的组，分组之后过滤数据。
having 后面，可以使用聚合函数。
```

1.7 分页查询

关键字：limit [offset,] rows

语法格式：

```
select * | 字段列表 [as 别名] from 表名
[where] 条件语句
[group by] 分组语句
[having] 过滤语句
[order by] 排序语句
[limit] 分页语句;

limit offset, length;
offset：开始行数，从 0 开始
length：每页显示的行数
```

limit 关键字不是 SQL92 标准提出的关键字，它是 MySQL 独有的语法。
通过 limit 关键字，MySQL 实现了物理分页。

分页分为逻辑分页和物理分页：

逻辑分页：将数据库中的数据查询到内存之后再进行分页。

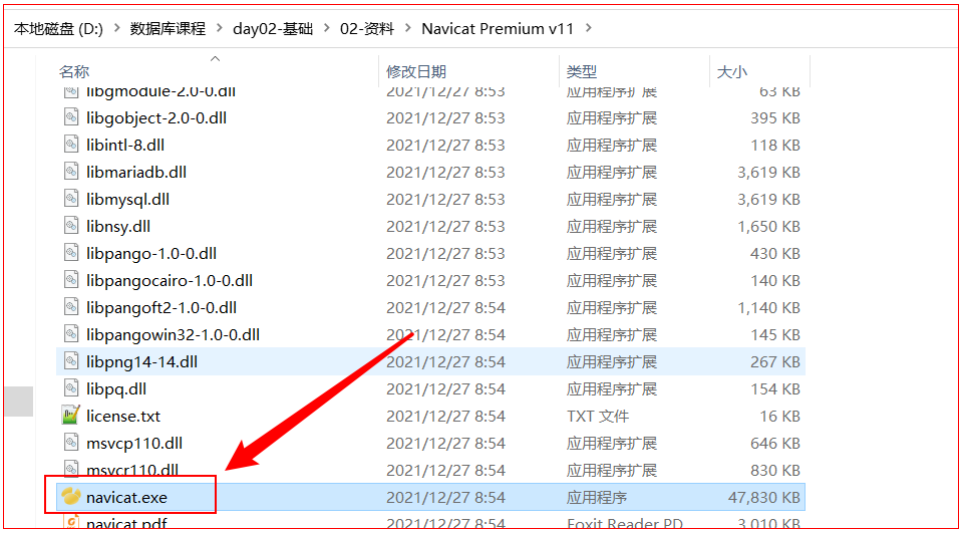
物理分页：通过 LIMIT 关键字，直接在数据库中进行分页，最终返回的数据，只是分页后的数据。

```
-- 如果省略第一个参数，默认从 0 开始
select * from product limit 5;
select * from product limit 3, 5;
```

第二章 MySQL图形化开发工具

2.1 安装Navicat【免安装】

提供的Navicat软件可直接使用



扩展：

1. Navicat
2. SQLyog
3. MySQL官方-MySQL workbench【监控功能强大】
4. IDEA 默认插件 -- 开发工具

2.2 使用Navicat

输入用户名、密码，点击连接按钮，进行访问MySQL数据库进行操作

> 编程大神：将复杂的问题，简单化
>
> 编程小白：将简单的问题，复杂化

第三章 数据库备份与恢复

3.1 备份

数据库的备份是指将数据库转换成对应的sql文件

1）MySQL命令备份

数据库导出sql脚本的格式：

```
mysqldump  -u用户名 -p密码 数据库名>生成的脚本文件路径
```

例如：

```
mysqldump  -uroot  -proot day02>d:\backup.sql
```

以上备份数据库的命令中需要用户名和密码，即表明该命令要在用户没有登录的情况下使用

2）可视化工具备份

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

3.2 恢复

数据库的恢复指的是使用备份产生的sql文件恢复数据库，即将sql文件中的sql语句执行就可以恢复数据库内容。

1）命令恢复

使用数据库命令备份的时候只是备份了数据库内容，产生的sql文件中没有创建数据库的sql语句，在恢复数据库之前需要自己动手创建数据库。

- 在数据库外恢复
  - **格式:** `mysql -uroot -p密码 数据库名 < 文件路径`
  - 例如: `mysql -uroot -proot day02<d:\backup.sql`
- 在数据库内恢复
  - **格式:** `source SQL脚本路径`
  - 例如: `source d:\backup.sql`
  - 注意:使用这种方式恢复数据，首先要登录数据库.

2) 可视化工具恢复

执行的SQL文件，执行即可。

第四章 多表操作

实际开发中，一个项目通常需要很多张表才能完成。

例如：一个**商城项目**就需要分类表(category)、商品表(products)、订单表(orders)等多张表。且这些表的数据之间存在一定的关系，接下来我们将在单表的基础上，一起学习多表方面的知识。

> - MySQL关系型数据库
> - 非关系型数据库：ES、Redis...

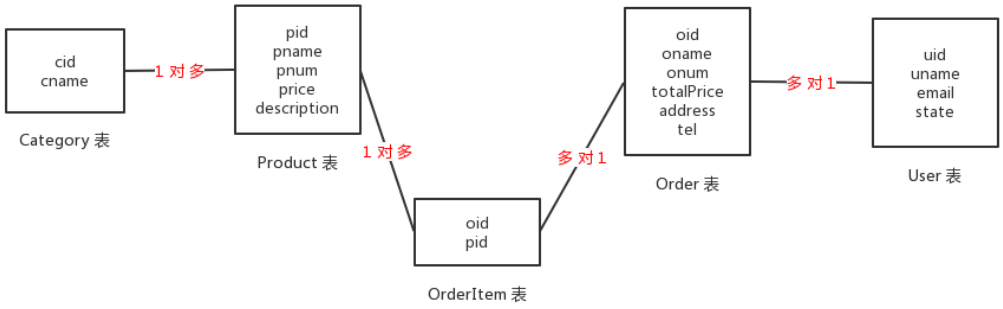
4.1 多表之间的关系

- 一读一：一夫一妻制度
- 一对多：一夫多妻制度
- 多对多：群居制度

表跟表之间的关系，大家可以理解为是实体跟实体的关系的一种映射。比如，导师与学员，订单与客户，部门与员工等等。

主要关系有三种：

```
一对一：比如，一个男的只能取一个女的当老婆。
一对多：比如，客户与订单，一个客户可以在商城中下多个订单。
多对多：比如，学生与课程，一个学校有很多学生，学生都可以学很多课程。
```



1）一对一关系

在实际工作中，一对一在开发中应用不多，因为一对一完全可以创建成一张表

建立两表的唯一一对一的关系：

- 第一步：将被关联的表主键唯一
- 第二步：建立外键约束，管理该表的唯一主键

案例：一个丈夫只能有一个妻子

```
CREATE TABLE wife(
    id  INT PRIMARY KEY ,
    wname  VARCHAR(20),
    sex CHAR(1)
);

CREATE TABLE husband(
    id  INT PRIMARY KEY ,
    hname  VARCHAR(20),
    sex CHAR(1)
);
```

外键唯一

一对一关系创建方式 1 之外键唯一：

添加外键列 wid，指定该列的约束为唯一（不加唯一约束就是一对多关系）

```
ALTER TABLE husband ADD wid INT  UNIQUE;
```

添加外键约束

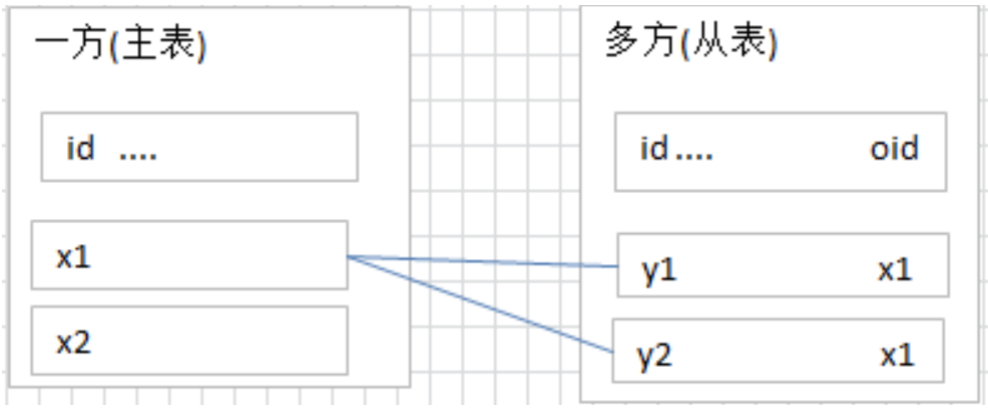
```
alter table husband add foreign key (wid) references wife(id);
```

主键做外键

一对一关系创建方式 2 之主键做外键：（大家下去自己练习）

思路：使用主表的主键作为外键去关联从表的主键

2）一对多关系



常见实例：一个分类对应多个商品，客户和订单，分类和商品，部门和员工.

总结：有外键的就是多的一方。

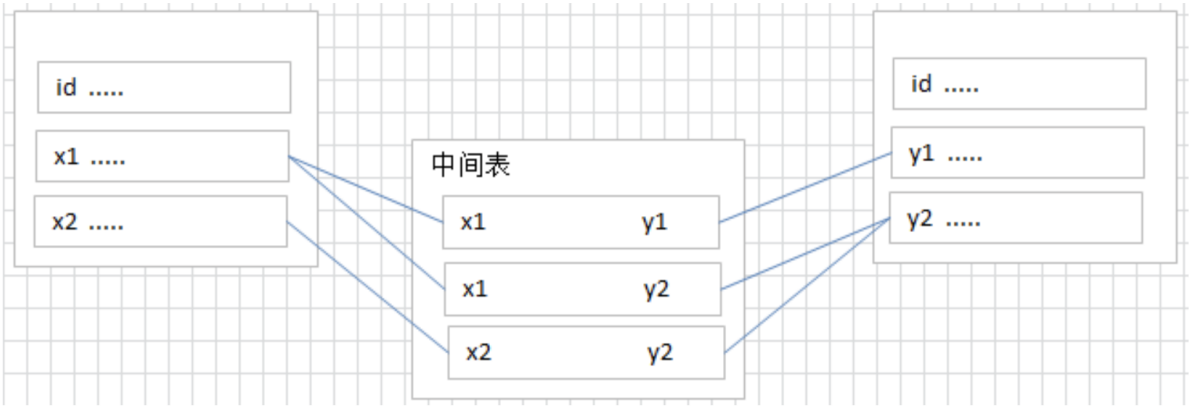
注意事项：一对多关系和一对一关系的创建很类似，唯一区别就是外键不唯一。

一对多关系创建：

- 添加外键列
- 添加外键约束

3）多对多关系

常见实例：学生和课程、用户和角色



注意事项：需要中间表去完成多对多关系的创建，多对多关系其实就是两个一对多关系的组合

多对多关系创建：

- 创建中间表，并在其中创建多对多关系中两张表的外键列

- 在中间表中添加外键约束
- 在中间表中添加联合主键约束

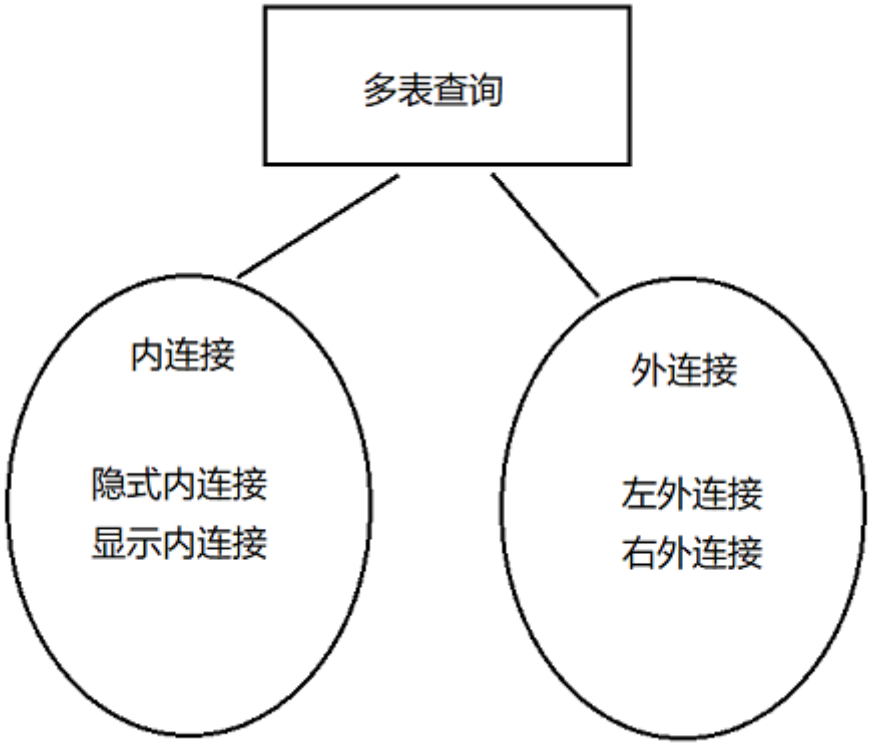
4.2 多表连接查询

4.2.1 多表查询的作用

比如，我们现在有一个员工，希望通过员工查询到对应的部门相关信息，部门名称、部门经理、部门收支等等。

员工 worker --- 部门 department

希望通过一条 SQL 语句进行查询多张相关的表，然后拿到的查询结果，其实是从多张表中综合而来的。比如，我们现在想获取张三的部门经理。



4.2.2 准备数据

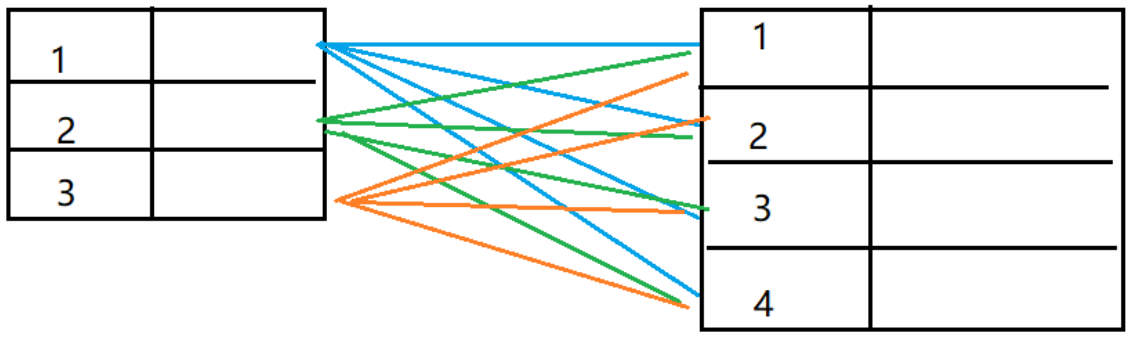
```sql
-- 部门表
create table department (
    id int primary key auto_increment,
    name varchar(50)
);
-- 插入部门数据
insert into department(name)
values('技术研发'), ('市场营销'), ('行政财务');

-- 员工表
create table worker (
    id int primary key auto_increment,
    name varchar(50),    -- 名字
    sex char(2),         -- 性别
    money double,        -- 工资
    inWork_date date,    -- 入职时间
    depart_id int,       -- 部门
```

```
    foreign key(depart_id) references department(id)
);
-- 插入员工数据
insert into worker(name, sex, money, inWork_date,depart_id)
values('cuihua', '女', 10000, '2019-5-5', 1);
insert into worker(name, sex, money, inWork_date,depart_id)
values('guoqing', '男', 20000, '2018-5-5', 2);
insert into worker(name, sex, money, inWork_date,depart_id)
values('qiangge', '男', 30000, '2018-7-5', 3);
insert into worker(name, sex, money, inWork_date,depart_id)
values('huahua', '女', 10000, '2019-5-5', 1);
```

4.2.3 笛卡尔集

```
-- 查询的时候
-- 左边表中的每一条记录和右边表的每一条记录都进行组合了
-- 出现的这种效果，就是笛卡尔集（但具体并不是我们希望得到的查询结果）
select * from worker, department;
```

1）如何产生



2）如何消除

通过上面的分析，我们知道有一些数据其实是无用的，只有满足 worker.depart_id = department.id 这个条件，过滤出来的数据才是我们想要的最终结果。

```
select * from worker, department where id = 3;

-- Column 'id' in where clause is ambiguous
-- select * from worker, department where id = 3;
-- 修改如下
select * from worker, department where worker.depart_id = department.id;
```

4.2.4 内连接

主要是使用左边的表中的记录去匹配右边表中的记录，如果满足条件的话，则显示查询结果。

```
从表.外键 = 主表.主键
```

1）隐式内连接（使用 where 关键字来指定条件）

```
select * from worker, department where worker.depart_id = department.id;
```

2）显示内连接（使用 inner..join..on 语句）

```
-- 1. 查询两张表
-- 使用 on 来指定条件
-- inner 关键字是可以省略的
select * from worker join department
on worker.depart_id = department.id ;

-- 2. 想查一个叫 cuihua 的人
-- 使用别名
select * from worker w join department d
on w.depart_id = d.id
where w.name = 'cuihua';

-- 3. 查询部分字段、
select w.id 员工编号, w.name 员工姓名, w.money 工资, d.name 部门名称 from worker
w join department d
on w.depart_id = d.id
where w.name = 'cuihua';
```

4.2.5 外连接

1）左外连接

使用 left outer join .. on

```
select 字段 from 左表 left outer join 右表 .. on 条件
```

```
-- 左外连接

-- 参考，内连接
select * from department d inner join worker w on d.id = w.depart_id;

-- 左外连接
-- outer 关键字是可以省略的
select * from department d left outer join worker w on d.id = w.depart_id;
```

当我们使用左边表中的记录去匹配右边表中的记录，如果满足条件的话，则显示；否则，显示为 null；
简单理解：在之前内连接的基础上，先保证左边的数据全部展示，再去找右表中的数据。

2）右外连接

使用 right outer join .. on

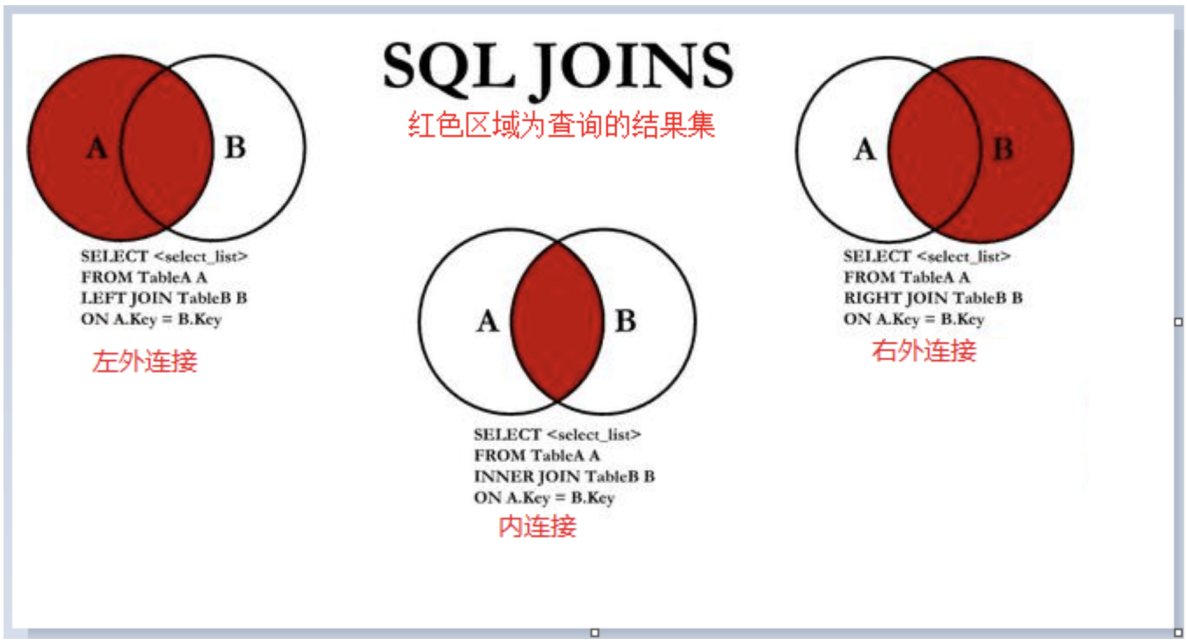
```
select 字段 from 左表 right outer join 右表.. on 条件
```

```
-- 右外连接
select * from department d right outer join worker w on d.id = w.depart_id;
```

先会使用右边的表中的记录去匹配左边表的记录，如果满足条件，则展示；否则，则显示 null；简单理解：在原来内连接的基础上，保证右边表中的全部数据都展示。

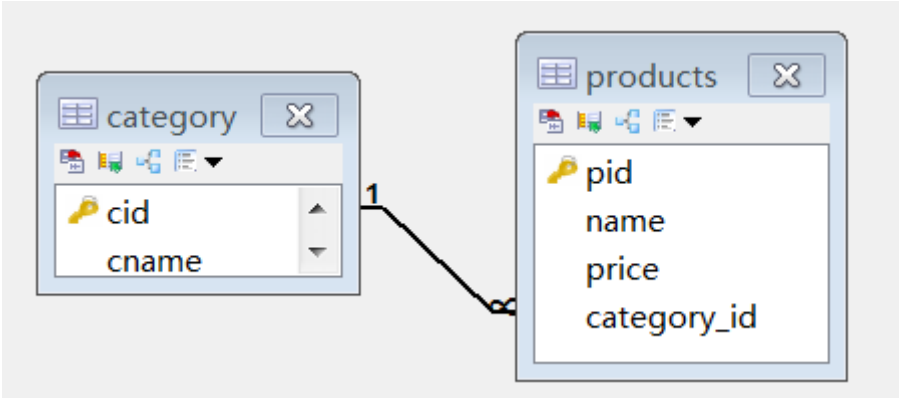
4.2.6 小结：连接区别

下面通过一张图说明连接的区别:



4.3 一对多操作案例

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

```sql
#创建分类表
create table category(
  cid varchar(32) PRIMARY KEY ,
  cname varchar(100) -- 分类名称
);

# 商品表
CREATE TABLE `products` (
  `pid` varchar(32) PRIMARY KEY  ,
  `name` VARCHAR(40) ,
  `price` DOUBLE
```

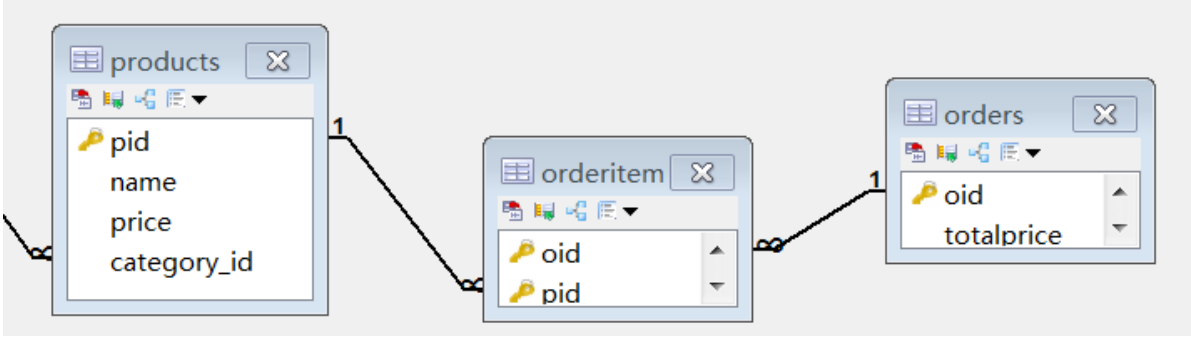
```
);

#添加外键字段
alter table products add column category_id varchar(32);

#添加约束
alter table products add constraint product_fk foreign key (category_id)
references category (cid);
```

操作

```
#1 向分类表中添加数据
INSERT INTO category (cid ,cname) VALUES('c001','服装');

#2 向商品表添加普通数据,没有外键数据，默认为null
INSERT INTO products (pid,pname) VALUES('p001','商品名称');

#3 向商品表添加普通数据，含有外键信息(category表中存在这条数据)
INSERT INTO products (pid ,pname ,category_id) VALUES('p002','商品名称
2','c001');

#4 向商品表添加普通数据，含有外键信息(category表中不存在这条数据) -- 失败,异常
INSERT INTO products (pid ,pname ,category_id) VALUES('p003','商品名称
2','c999');

#5 删除指定分类(分类被商品使用) -- 执行异常
DELETE FROM category WHERE cid = 'c001';
```

4.4 多对多操作案例

分析



- 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

实现：订单和商品

```
#商品表[已存在]

#订单表
create table `orders`(
  `oid` varchar(32) PRIMARY KEY ,
  `totalprice` double   #总计
);
```

```
8
9  #订单项表
10 create table orderitem(
11   oid varchar(50),-- 订单id
12   pid varchar(50)-- 商品id
13 );
14
15 #订单表和订单项表的主外键关系
16 alter table `orderitem` add constraint orderitem_orders_fk foreign key (oid)
   references orders(oid);
17
18 #商品表和订单项表的主外键关系
19 alter table `orderitem` add constraint orderitem_product_fk foreign key
   (pid) references products(pid);
20
21 #联合主键（可省略）
22 alter table `orderitem` add primary key (oid,pid);
```

操作

```
1  #1 向商品表中添加数据
2  INSERT INTO products (pid,pname) VALUES('p003','商品名称');
3
4  #2 向订单表中添加数据
5  INSERT INTO orders (oid ,totalprice) VALUES('x001','998');
6  INSERT INTO orders (oid ,totalprice) VALUES('x002','100');
7
8  #3向中间表添加数据(数据存在)
9  INSERT INTO orderitem(pid,oid) VALUES('p001','x001');
10 INSERT INTO orderitem(pid,oid) VALUES('p001','x002');
11 INSERT INTO orderitem(pid,oid) VALUES('p002','x002');
12
13 #4删除中间表的数据
14 DELETE FROM orderitem WHERE pid='p002' AND oid = 'x002';
15
16 #5向中间表添加数据(数据不存在) -- 执行异常
17 INSERT INTO orderitem(pid,oid) VALUES('p002','x003');
18
19 #6删除商品表的数据 -- 执行异常
20 DELETE FROM products WHERE pid = 'p001';
```