# 连接查询

## 基本概念

就是将两个或两个以上的表，“连接起来”，当做一个数据源，并从中去取得所需要的数据；

到底怎么连接？

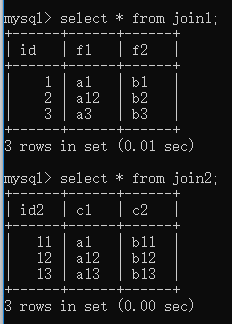
将每一个表的每一行数据两两之间互相对接起来，每次对接的结果的“一行”数据

举例：

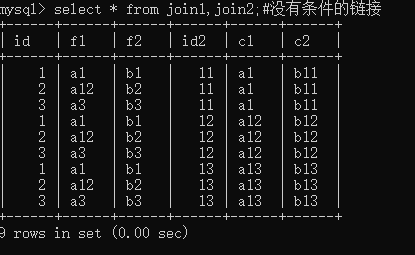
Source语句 导入

Source 路径

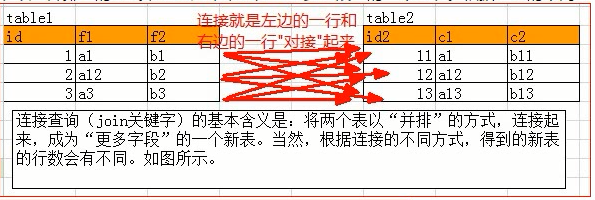
先看原始数据



然后将他们进行“连接”：



他们其实是这样的方式连接起来的结果



这种没有条件的连接，也可以写成这样：（交叉连接）

Select \* from 表1，表2

Select \* from 表1 join 表2

Select \* from 表1 cross join表2

其实，他们都叫做“交叉连接”，其实就是没有条件的连接——全部连接起来

## 连接基本形式

在代码级别，连接的基本形式为：

表1 【连接形式】join 表2【on 连接条件】；

如果是三个表，也可以这样，则进一步扩展为

表1 【连接形式】join 表2【on 连接条件】【连接形式】join 表3【on 连接条件】

更多表，以此类推

# 连接分类

## 主要分为以下连接：

## 交叉连接cross join：

就是刚才基本概念的时候的连接形式（结果）——它没有条件，只是按连接的基本概念，将所有数据行都连接起来的结果，它又叫“笛卡尔积”

对于表1（n1个字段，n2行），表2（m1个字段，m2行），他们交叉连接的结果是：

有n1+m1个列

有n2\*m2个行

形式有：

Select \* from 表1，表2；

Select \* from 表1join 表2；

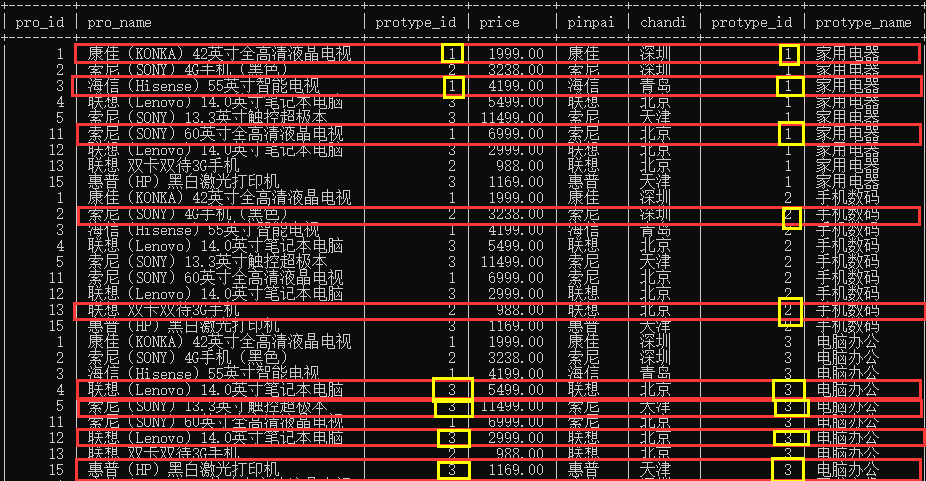
Select \* from 表1 cross join表2；

Select \* from 表1 inner join表2；

## 内连接：

形式有：

Select \* from 表1【inner】 join表2 on 连接条件；



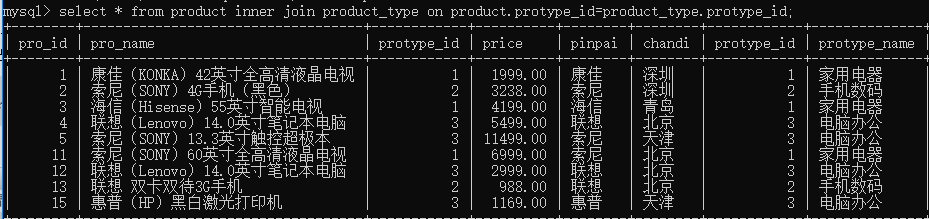
框中框框内的“数据行”，是正确的，有意义的数据，而其他行就是错误的数据

这些有意义的数据的一个特点，就是，所标示出的两个字段的值相等

该两个字段，分别来源于两个参与连接的表

我们的内连接，就是要找出这样的一些有意义的数据

如下：



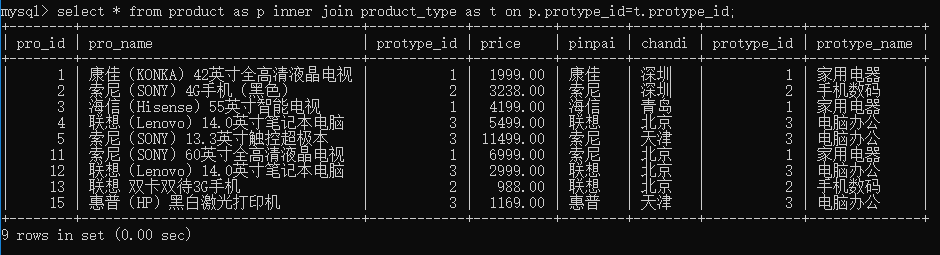
可见，这里查询的结果，都是前述查询（交叉连接）中，有意义的数据，其中：product.protype\_id=product\_type.protype\_id

被称为连接条件，他基本上就是我们之前所学的“外键关系”的一个描述。

注意

这种表跟表之间的内连接查询，虽然可以提现为表跟表支架你的关系——通常就是外键关系，但并不是有外键关系才能使用。

另外的一些变通形式：





## 左（外）连接left（outer）join：

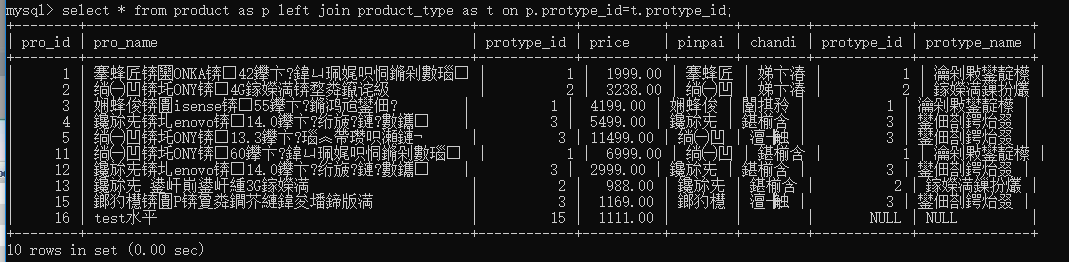
形式：

表1（左表）left 【outer】 join表2（右表）on条件

含义：

其实就是将两个表的内连接的结果，在“加上”左边表的不符合内连接所设定的条件的那些数据的结果；

举例：



可见，左连接的结果，左边的数据，一定都会全部取出来

## 右（外）连接right（outer）join：

形式：

表1（左表）right 【outer】 join表2（右表）on条件

含义：

其实就是将两个表的内连接的结果，在“加上”右边表的不符合内连接所设定的条件的那些数据的结果；

举例



## 全（外）连接full（outer）join

形式：

没有形式，因为mysql不支持全连接的语法

含义：

其实就是将两个表的内连接的结果，在“加上”左边表的不符合内连接所设定的条件的那些数据的结果；再“加上”右边表的不符合内连接所设定的条件的那些数据的结果；

简单的话，左边和右边加一起

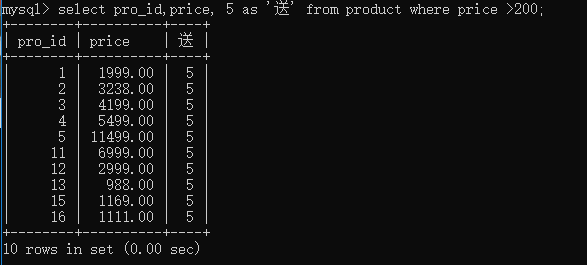
# 子查询

## 基本含义

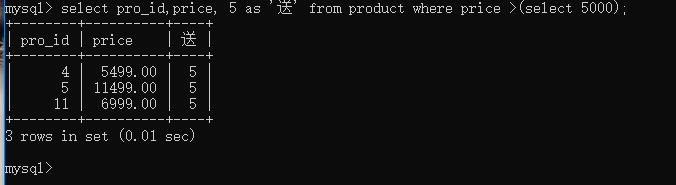
一个select语句，就是一个查询语句；

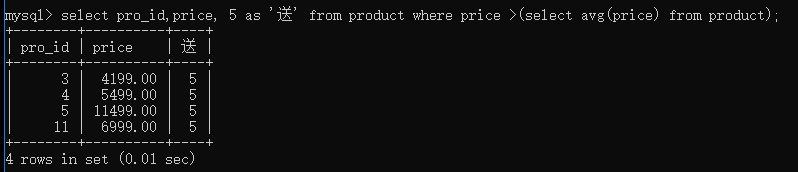
Select字段或表达式 from 数据源 where XX 条件判断 ；

上述select部分，from部分，where部分，往往都是一些“数据”或数据的组合；比如：



From部分，当然就是“表”，或“表的连接结果”，他们也是数据，只是通常为“表数据”





这就是子查询，其结果是：找出所有商品的平均价，则，该总的select语句的含义就有了找出所有大于平均值的商品

可见，所谓的子查询，就是在一个查询语句（select）中的内部，某些位置，又出现的“查询语句”。

则有两个概念：

主查询：

子查询

通常，子查询是为主查询服务的，而，通常，都是子查询获得一定的结果数据之后1，才去执行主查询；

在形式上，可以有如下表达：

* **selelct 字段或表达式或子查询 [as 别名] from 表名或链接结果或子查询 where 字段或表达式或子查询的条件判断**

即可以在这几个位置出现子查询（其中having其实也可以，因为它跟where是一样含义）；

## 子查询分类

按子查询结果，分为：

* 表子查询 ： 一个子查询返回的结果理论上是“多行多列”的时候。此时可以当做一个“表”来使用，通常是放在from后面。
* 行字查询 ： 一个子查询返回的结果理论上是“一行多列”的时候。此时可以当做一个“行”来使用，通常放在“行比较语法”中。
* 行比较语法类似这样：where row（字段1，字段2）=（select行子查询）
* 列子查询 ： 一个子查询返回的结果理论上是“多行一列”的时候。此时可以当做“多个值”使用，类似这种：(5, 17, 8, 22)。
* 标量子查询：一个子查询返回的结果理论上是“一行一列”的时候。此时可以当做“一个值”使用，类似这种：select 5 as c1; 或select ...where a = 17，或select ... where b > 8;即上述“单个数据”可以用标量来代替

子查询，按位置（场合）分：

* 作为主查询的结果数据：select c1,**(select f1 from tab2) as f11** from tab1; #这里子查询应该只有一个数据（一行一列，标量子查询）
* 作为主查询的条件数据：select c1 from tab1 where c1 in **(select f1 from tab2)**; #这里子查询可以是多个数据（多行一列，列子查询）
* 作为主查询的来源数据：select c1 from **(select f1 as c1, f2 from tab2) as t2**; #这里子查询可以是任意查询结果（表子查询）。

## 常见子查询

## 比较运算符中的子查询

形式为：

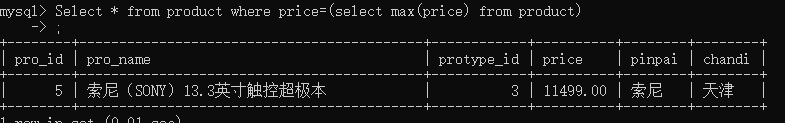
操作数 比较运算符（标量子查询）；

说明：

操作数，其实就是比较运算符的2个数据之一而已，通常就是一个字段名；

举例1：找出最高的商品

Select \* from product where price=(select max(price) from product)



## 使用in的子查询

以前用的in的用法：

Xx in （值1，值2….）

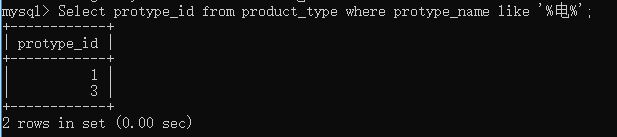
则in子查询为：

Xx in（列子查询）

举例2：查出所有类别中带“电”这个字的产品

可以分两步思考

Select protype\_id from product\_type where protype\_name like ‘%电%’;



然后，将此结果，当做“in”的数据项来使用，去找出商品

Select \* from product where protype\_id in (

Select protype\_id from product\_type where protype\_name like ‘%电%’

));

代码运行出错 待解决

## 使用any子查询（some）

形式：

操作数 比较运算符 any（列子查询）列！！

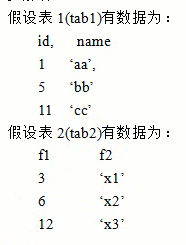
含义：

当某个操作数（字段）对于该列子查询的其中任一个值，都满足该比较运算符，则就算是满足了条件；

即只要有一个值满足就算是满足

进一步解释

假设表一有数据为：



Select \* from tab1 where id > any (select f1 from tab2)

则可以取出的结果数据有：

表tab1 中的5和11

## 使用all的子查询

形式：

操作数 比较运算符 any（列子查询）列！！

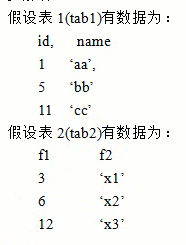
含义：

当某个操作数（字段）对于该列子查询的所有值，才满足该比较运算符，才算是满足了条件；

即全部值都满足就算是满足

进一步解释

假设表一有数据为：



Select \* from tab2 where f1> all (select id from tab1 )；

则可以取出的结果数据有：

结果只有12满足

举例1：

查询出所有非最高价的商品

第一步思考

Select price from product

第二步思考,这些非最高价的商品的价格

只是会小于上述“所有价格”中的某一个

Select \* from product where price < any(Select price from product)

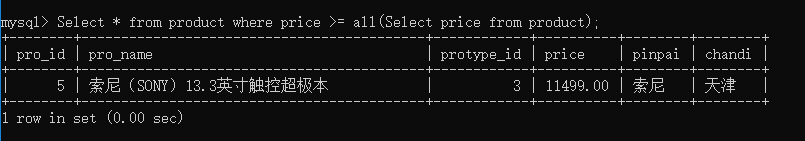


举例2：

查询出所有最高价的商品

思考：最高价的商品价格，有一个特征，大于等于所有价格

Select \* from product where price >= all(Select price from product)



## 使用some的子查询

Some是any的同义词。

## 使用exists的查询

形式：

Where exist（子查询）

含义：

该子查询如果有“数据”，则exists的结果就是true，否则就是false

说明：

因为，exists子查询的该含义，造成主查询往往出现这样的情形：要么全部取出，要么都不取出

如果局限于这个含义（使用情形），其基本就失去了他的现实使用含义。

但：

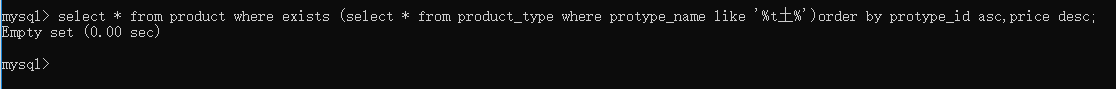
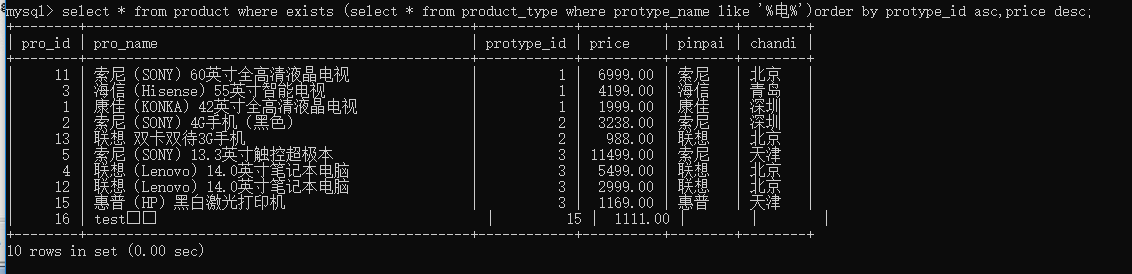
实际应用中，该子查询，往往都不是独立的子查询，而是会需要跟“主查询”的数据源（表），建立某种关系——通常就是连接关系。建立的方式是“隐式的”，即没有在代码上体现关系，但却在内部有1其连接的“实质”。

此隐式方式，通常就体现在子查询中的where条件语句中，使用主查询表中的数据（字段）；

举例：

查询商品信息表其类别名称中带“电”这个字的所有商品；

select \* from product where exists (select \* from product\_type where protype\_name like '%电%')order by protype\_id asc,price desc;



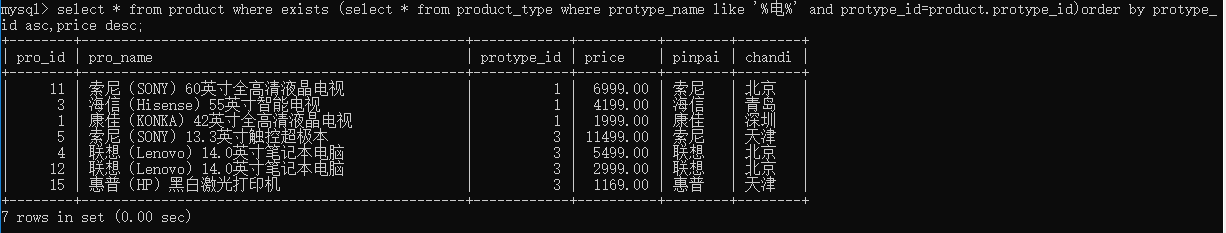
select \* from product where exists (select \* from product\_type where protype\_name like '%电%' and protype\_id=product.protype\_id)order by protype\_id asc,price desc;【隐含的连接条件】

注意：

1这种子查询语句，没法“独立存在（独立运行）”，而是必须跟主查询一起使用，

2其他所学的子查询，是可以独立运行的，而且会得到一个运行的结果

3.该子查询中的条件，应该设定为跟主查询的某个字段有一定的关联性判断，通常该判断就是这两个表的“本来该有的连接条件”



结论：

如果一个查询需求，可以使用连接查询的，也可以使用子查询得到，则通常推荐使用连接查询，效率更高

## 联合查询union

一个翻译问题的解释：

在mysql的手册中，将连接查询（join）翻译为联合查询；

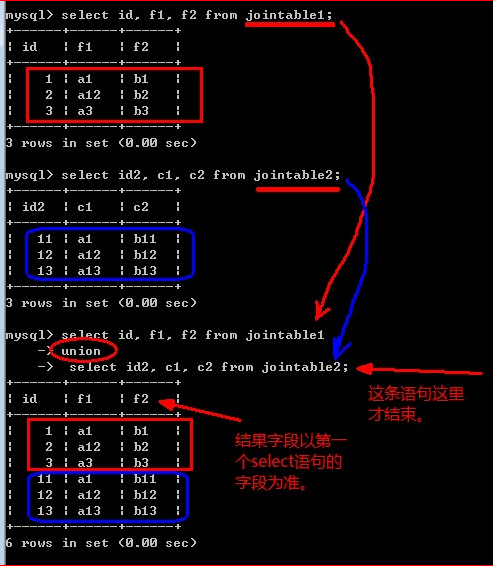
而联合查询（union），没有明确翻译；

但：

在通常的书籍或文章，join被翻译为“连接”查询，而union才翻译为联合查询。

## 基本概念：

将两个具有相同字段数量的查询语句的结果，以“上下堆叠”的方式，合并为一个查询结果。



可见：

* + - 1. 两个select语句的查询结果的“字段书”必须一致
      2. 通常，也应该让两个查询语句的字段类型具有一致性
      3. 也可以联合更多的查询结果

## 语法形式

Select语句1

Union【all | distinct】

Select语句2

此联合查询语句，默认会“自动消除修复行”，即默认是distinct

如果想要将所有数据都显示（允许重复行），就使用all

即，这里写all才有意义：

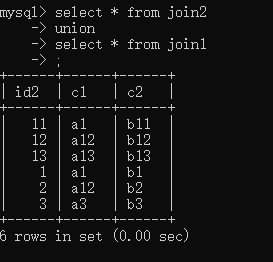
对比

Select 【all | distinct】…….

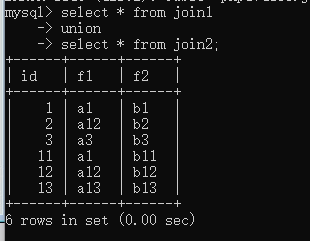
对于select语句，写distinct才有意义

## 细节

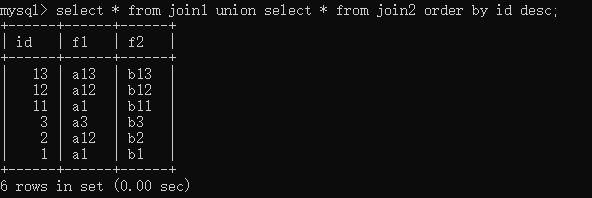
* 两个select语句的查询结果的字段需要保持一致：个数必须相同，对应顺序上的字段类型也应该相同。



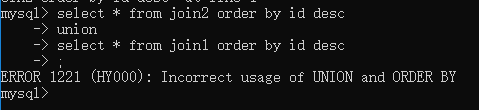
对比



* 如果要对整个查询结果进行order by或limit操作，则需要将两个select语句用括号括起来，类似这样：(select...)union(select...) order by XXX limit m,n;

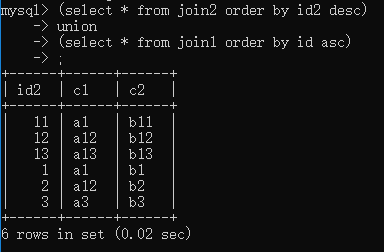
基本用法

不对的做法：

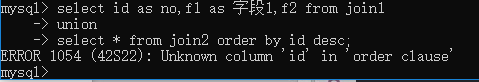


错误：不正确的使用了union和order by

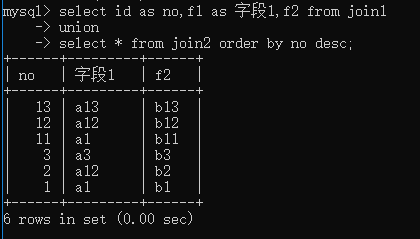
无效的做法



* 如果第一个select语句中的列有别名，则order by子句中就必须使用该别名。



修改



实现全外连接

