子查询与表连接

讲师： 伊川

子查询(嵌套sql)

> SELECT语句是SQL的查询。迄今为止我们所看到的所有SELECT语句都是简单查询，即从单个数据库表中检索数据的单条语句。
>
> SQL还允许创建子查询（subquery），即嵌套在其他查询中的查询。

利用子查询进行过滤

订单存储在两个表中。对于包含订单号、客户ID、订单日期的每个订单，orders表存储一行。 各订单的物品存储在相关的orderitems表中。orders表不存储客户信息。它只存储客户的ID。

实际的客户信息存储在customers表中。

现在，假如需要列出订购物品TNT2的所有客户，应该怎样检索?

```
--(1) 检索包含物品TNT2的所有订单的编号。
 select order_num from orderitems where prod_id = 'TNT2';
+-----------+
| order_num |
+-----------+
|     20005 |
|     20007 |
+-----------+
--(2) 检索具有前一步骤列出的订单编号的所有客户的ID
 select cust_id from orders where order_num IN (20005,20007);
+---------+
| cust_id |
+---------+
|   10001 |
|   10004 |
+---------+
--(3) 检索前一步骤返回的所有客户ID的客户信息。
 select cust_name,cust_contact from customers where cust_id in (10001,10004);
+----------------+--------------+
| cust_name      | cust_contact |
+----------------+--------------+
| Coyote Inc.    | Y Lee        |
| Yosemite Place | Y Sam        |
+----------------+--------------+
```

可以把其中的WHERE子句转换为子查询而不是硬编码这些SQL返回的数据：

```
 select cust_name,cust_contact
 from customers
 where cust_id in (select cust_id
          from orders
          where order_num IN (select order_num
                    from orderitems
                    where prod_id = 'TNT2'));
+----------------+--------------+
| cust_name      | cust_contact |
+----------------+--------------+
| Coyote Inc.    | Y Lee        |
| Yosemite Place | Y Sam        |
+----------------+--------------+
--为了执行上述SELECT语句，MySQL实际上必须执行3条SELECT语句。
--最里边的子查询返回订单号列表，此列表用于其外面的子查询的WHERE子句。
--外面的子查询返回客户ID列表，此客户ID列表用于最外层查询的WHERE子句。
--最外层查询确实返回所需的数据。
```

> 这里给出的代码有效并获得所需的结果。
>
> > 但是，使用子查询并不总是执行这种类型的数据检索的最有效的方法。

作为计算字段使用子查询

> 使用子查询的另一方法是创建计算字段。

```
-- 假如需要显示customers表中每个客户的订单总数。订单与相应的客户ID存储在orders表中。

-- (1) 从customers表中检索客户列表。
 select cust_id,cust_name from customers ;
+---------+----------------+
| cust_id | cust_name      |
+---------+----------------+
|   10001 | Coyote Inc.    |
|   10002 | Mouse House    |
|   10003 | Wascals        |
|   10004 | Yosemite Place |
|   10005 | E Fudd         |
+---------+----------------+

-- (2) 对于检索出的每个客户，统计其在orders表中的订单数目。
 select count(*) as orders  from orders  where cust_id = 10001;
+--------+
| orders |
+--------+
|      2 |
+--------+
```

> 为了对每个客户执行COUNT()*计算，应该将COUNT()*作为一个子查询。

```
select cust_id,cust_name,
    (select count(*)
          from orders
          where orders.cust_id = customers.cust_id) as orders
         )
from customers
order by cust_name;
+---------+---------------+--------+
| cust_id | cust_name     | orders |
+---------+---------------+--------+
|   10001 | Coyote Inc.   |      2 |
|   10005 | E Fudd        |      1 |
|   10002 | Mouse House   |      0 |
|   10003 | Wascals       |      6 |
|   10004 | Yosemite Place |      1 |
+---------+---------------+--------+
```

> orders是一个计算字段，它是由圆括号中的子查询建立的。该子查询对检索出的每个客户执行一次。在此例子中，该子查询执行了5次，因为检索出了5个客户。
>
> > 注意:子查询中的WHERE子句与前面使用的WHERE子句稍有不同，因为它使用了完全限定列名这种类型的子查询称为相关子查询。任何时候只要列名可能有多义性，就必须使用这种语法（表名和列名由一个句点分隔）。因为有两个cust_id列，一个在customers中，另一个在orders中，需要比较这两个列以正确地把订单与它们相应的顾客匹配。如果不完全限定列名，MySQL将假定你是对orders表中的cust_id进行自身比较。

关系表

> SQL最强大的功能之一就是能在数据检索查询的执行中联结（join）表。
>
> 在能够有效地使用联结前，必须了解关系表以及关系数据库设计的一些基础知识。

```
--假如有一个包含产品目录的数据库表，其中每种类别的物品占一行。
--对于每种物品要存储的信息包括产品描述和价格，以及生产该产品的供应商信息。
产品表:
  产品，描述，价格，供应商名称，供应商地址,供应商联系方式
  A6    ...  ...  奥迪     ...  ....
  520li .. .... 宝马    ... ...
  ...

--现在，假如有由同一供应商生产的多种物品，那么在何处存储供应
--商信息（如，供应商名、地址、联系方法等）呢?
  产品，描述，价格，供应商名称，供应商地址,供应商联系方式
  A6    ...  ...  奥迪     ...  ....
  520li .. .... 宝马    ... ...
```

```
  A8    ..    ...  奥迪     ...    ...
```

相同数据出现多次决不是一件好事，此因素是关系数据库设计的基础。

关系表的设计就是要保证把信息分解成多个表，一类数据一个表。

各表通过某些常用的值（即关系设计中的关系（relational））互相关联。

在这个例子中，可建立两个表，一个存储供应商信息，另一个存储产品信息。

```
-- vendors表包含所有供应商信息
|vend_id | vend_name | vend_address| vend_city ....

-- products表只存储产品信息，它除了存储供应商ID（vendors表的主键）外不存储其他供应商信息。
prod_id | vend_id | prod_name | prod_price | prod_desc
```

> vendors表的主键又叫作products的外键，它将vendors表与products表关联，利用供应商ID能从vendors表中找出相应供应商的详细信息。 这样做的好处如下：
>
> - **供应商信息不重复，从而不浪费时间和空间；**
> - **如果供应商信息变动，可以只更新vendors表中的单个记录，相关表中的数据不用改动；**
> - **由于数据无重复，显然数据是一致的，这使得处理数据更简单**
>
> 关系数据可以有效地存储和方便地处理。因此，关系数据库的可伸缩性远比非关系数据库要好。

表联结

> 如果数据存储在多个表中，怎样用单条SELECT语句检索出数据?
>
> 答案是使用联结。简单地说，联结是一种机制，用来在一条SELECT语句中关联表，因此称之为联结。
>
> 使用特殊的语法，可以联结多个表返回一组输出，联结在运行时关联表中正确的行。

例如：我们需要查询出所有的商品及对应的供应商信息怎么办?

```
-- 联结的创建非常简单，规定要联结的所有表以及它们如何关联即可。
select vend_name,prod_name,prod_price
from vendors,products
where vendors.vend_id = products.vend_id
order by vend_name,prod_name;
+-------------+----------------+------------+
| vend_name   | prod_name      | prod_price |
+-------------+----------------+------------+
| ACME        | Bird seed      |      10.00 |
| ACME        | Carrots        |       2.50 |
| ACME        | Detonator      |      13.00 |
| ACME        | Safe           |      50.00 |
| ACME        | Sling          |       4.49 |
| ACME        | TNT (1 stick)  |       2.50 |
```

```
| ACME         | TNT (5 sticks) |      10.00 |
| Anvils R Us  | .5 ton anvil   |       5.99 |
| Anvils R Us  | 1 ton anvil    |       9.99 |
| Anvils R Us  | 2 ton anvil    |      14.99 |
| Jet Set      | JetPack 1000   |      35.00 |
| Jet Set      | JetPack 2000   |      55.00 |
| LT Supplies  | Fuses          |       3.42 |
| LT Supplies  | Oil can        |       8.99 |
+--------------+----------------+------------+
14 rows in set (0.00 sec)

--这两个表用WHERE子句正确联结，WHERE子句指示MySQL匹配vendors表中的vend_id和products表中的vend_id。
--可以看到要匹配的两个列以 vendors.vend_id 和 products. vend_id指定。这里需要这种完全限定列名，因为如果只给出vend_id，则MySQL不知道指的是哪一个（它们有两个，每个表中一个）。
--在引用的列可能出现二义性时，必须使用完全限定列名（用一个点分隔的表名和列名）。
```

> 在联结两个表时，你实际上做的是将第一个表中的每一行与第二个表中的每一行配对。
>
> WHERE子句作为过滤条件，它只包含那些匹配给定条件（这里是联结条件）的行。
>
> 你能想象上面的sql如果没有where条件时会怎样吗?

`select vend_name,prod_name,prod_price from vendors,products`

> **如果没有where条件，第一个表中的每个行将与第二个表中的每个行配对，而不管它们逻辑上是否可以配在一起**
>
> **由没有联结条件的表关系返回的结果为笛卡儿积。检索出的行的数目将是第一个表中的行数乘以第二个表中的行数。**

不要忘了WHERE子句

> 应该保证所有联结都有WHERE子句，否则MySQL将返回比想要的数据多得多的数据。
>
> 同理，应该保证WHERE子句的正确性。不正确的过滤条件将导致MySQL返回不正确的数据

其实，对于这种联结可以使用稍微不同的语法来明确指定联结的类型。

`select vend_name,prod_name,prod_price from vendors inner join products on vendors.vend_id = products.vend_id;`

> 两个表之间的关系是FROM子句的组成部分，以INNER JOIN指定。
>
> 在使用这种语法时，联结条件用特定的ON子句而不是WHERE子句给出。
>
> 传递给ON的实际条件与传递给WHERE的相同。
>
> SQL规范首选INNER JOIN语法。

联结多个表

> SQL对一条SELECT语句中可以联结的表的数目没有限制。

> 创建联结的基本规则也相同。首先列出所有表，然后定义表之间的关系。

```
select prod_name,vend_name,prod_price,quantity
from orderitems,products,vendors
  where products.vend_id = vendors.vend_id
  and orderitems.prod_id = products.prod_id
  and order_num = 20005;
```

> MySQL在运行时关联指定的每个表以处理联结。 这种处理可能是非常耗费资源的，因此应该仔细，不要联结不必要的表。联结的表越多，性能下降越厉害。

使用表别名 AS

> 别名除了用于列名和计算字段外，SQL还允许给表名起别名。
>
> 这样做有两个主要理由：
>
> - 缩短SQL语句；
> - 允许在单条SELECT语句中多次使用相同的表
>
> 应该注意，表别名只在查询执行中使用。与列别名不一样，表别名不返回到客户机

自联结

> 假如你发现某物品（其ID为DTNTR）存在问题，因此想知道生产该物品的供应商生产的其他物品是否也存在这些问题。此查询要求首先找到生产ID为DTNTR的物品的供应商，然后找出这个供应商生产的其他物品。

```
 -- 使用子查询(嵌套查询)
 select prod_id,prod_name
 from products
  where vend_id = (select vend_id from products where prod_id = 'DTNTR');
+---------+----------------+
| prod_id | prod_name      |
+---------+----------------+
| DTNTR   | Detonator      |
| FB      | Bird seed      |
| FC      | Carrots        |
| SAFE    | Safe           |
| SLING   | Sling          |
| TNT1    | TNT (1 stick)  |
| TNT2    | TNT (5 sticks) |
+---------+----------------+

-- 使用联结的相同查询：
 select p1.prod_id,p1.prod_name
 from products as p1,products as p2
  where p1.vend_id = p2.vend_id and p2.prod_id = 'DTNTR';
```

```
+---------+----------------+
| prod_id | prod_name      |
+---------+----------------+
| DTNTR   | Detonator      |
| FB      | Bird seed      |
| FC      | Carrots        |
| SAFE    | Safe           |
| SLING   | Sling          |
| TNT1    | TNT (1 stick)  |
| TNT2    | TNT (5 sticks) |
+---------+----------------+
-- 此查询中需要的两个表实际上是相同的表，因此products表在FROM子句中出现了两次。虽然这是完全合法的，但对products的引用具有二义性，因为MySQL不知道你引用的是products表中的哪个实例。
-- 为解决此问题，使用了表别名。products的第一次出现为别名p1，第二次出现为别名p2。现在可以将这些别名用作表名。
--例如，SELECT语句使用p1前缀明确地给出所需列的全名。如果不这样，MySQL将返回错误，因为分别存在两个名为prod_id、prod_name的列。MySQL不知道想要的是哪一个列（即使它们事实上是同一个列）。WHERE（通过匹配p1中的vend_id和p2中的vend_id）首先联结两个表，然后按第二个表中的prod_id过滤数据，返回所需的数据
```

> 用自联结而不用子查询 自联结通常作为外部语句用来替代从相同表中检索数据时使用的子查询语句。
>
> 虽然最终的结果是相同的，但有时候处理联结远比处理子查询快得多。

外部链接

> 许多联结将一个表中的行与另一个表中的行相关联。但有时候会需要包含没有关联行的那些行。
>
> 例如，可能需要使用联结来完成以下工作：

- 对每个客户下了多少订单进行计数，包括那些至今尚未下订单的客户；
- 列出所有产品以及订购数量，包括没有人订购的产品；
    - 计算平均销售规模，包括那些至今尚未下订单的客户

> 在上述例子中，联结包含了那些在相关表中没有关联行的行。这种类型的联结称为外部联结。

```
-- 内部联结。它检索所有客户及其订单：
select customers.cust_id,orders.order_num from customers inner join orders on customers.cust_id = orders.cust_id;
+---------+-----------+
| cust_id | order_num |
+---------+-----------+
|   10001 |     20005 |
|   10001 |     20009 |
|   10003 |     20006 |
|   10004 |     20007 |
|   10005 |     20008 |
+---------+-----------+
5 rows in set (0.00 sec)
```

```
--外部联结语法类似。检索所有客户，包括那些没有订单的客户
 select customers.cust_id,orders.order_num from customers left join orders on
customers.cust_id = orders.cust_id;
+---------+-----------+
| cust_id | order_num |
+---------+-----------+
|   10001 |     20005 |
|   10001 |     20009 |
|   10002 |      NULL |
|   10003 |     20006 |
|   10004 |     20007 |
|   10005 |     20008 |
+---------+-----------+
6 rows in set (0.00 sec)
```

聚集函数也可以方便地与其他联结一起使用。

如果要检索所有客户及每个客户所下的订单数，下面使用了COUNT()函数的代码可完成此工作

包含那些没有任何下订单的客户。

```
select customers.cust_name,customers.cust_id,count(orders.order_num) as
num_ord from customers left join orders on customers.cust_id = orders.cust_id
group by customers.cust_id;
+----------------+---------+---------+
| cust_name      | cust_id | num_ord |
+----------------+---------+---------+
| Coyote Inc.    |   10001 |       2 |
| Mouse House    |   10002 |       0 |
| Wascals        |   10003 |       1 |
| Yosemite Place |   10004 |       1 |
| E Fudd         |   10005 |       1 |
+----------------+---------+---------+
```

> - 保证使用正确的联结条件，否则将返回不正确的数据。
> - 应该总是提供联结条件，否则会得出笛卡儿积。
> - 在一个联结中可以包含多个表，甚至对于每个联结可以采用不同的联结类型。虽然这样做是合法的，一般也很有用，但应该在一起测试它们前，分别测试每个联结。这将使故障排除更为简单

组合查询 UNION

> MySQL也允许执行多个查询（多条SELECT语句），并将结果作为单个查询结果集返回。
>
> 这些组合查询通常称为并（union）或复合查询（compound query）。
>
> ### UNION规则
>
> - UNION必须由两条或两条以上的SELECT语句组成，语句之间用关键字UNION分隔（因此，如果组合4条SELECT语句，将要使用3个UNION关键字）。
> - UNION中的每个查询必须包含相同的列、表达式或聚集函数（不过各个列不需要以相同的次序列出）
> - 列数据类型必须兼容：类型不必完全相同，但必须是DBMS可以隐含地转换的类型（例如，不同的数值类型或不同的日期类型）。

```
--假如需要价格小于等于5的所有物品的一个列表，而且还想包括供应商1001和1002生产的所有物品。

-- 先查询第一个结果
 select vend_id,prod_id,prod_price from products where prod_price <= 5;
+---------+---------+------------+
| vend_id | prod_id | prod_price |
+---------+---------+------------+
|    1003 | FC      |       2.50 |
|    1002 | FU1     |       3.42 |
|    1003 | SLING   |       4.49 |
|    1003 | TNT1    |       2.50 |
+---------+---------+------------+
4 rows in set (0.00 sec)

-- 再查询第二个结果
 select vend_id,prod_id,prod_price from products where vend_id in(1001,1002);
+---------+---------+------------+
| vend_id | prod_id | prod_price |
+---------+---------+------------+
|    1001 | ANV01   |       5.99 |
|    1001 | ANV02   |       9.99 |
|    1001 | ANV03   |      14.99 |
|    1002 | FU1     |       3.42 |
|    1002 | OL1     |       8.99 |
+---------+---------+------------+
5 rows in set (0.00 sec)


--使用union将两个sql一并执行
 select vend_id,prod_id,prod_price from products where prod_price <= 5
 union
 select vend_id,prod_id,prod_price from products where vend_id in(1001,1002);
+---------+---------+------------+
| vend_id | prod_id | prod_price |
+---------+---------+------------+
|    1003 | FC      |       2.50 |
```

```
|    1002 | FU1      |       3.42 |
|    1003 | SLING    |       4.49 |
|    1003 | TNT1     |       2.50 |
|    1001 | ANV01    |       5.99 |
|    1001 | ANV02    |       9.99 |
|    1001 | ANV03    |      14.99 |
|    1002 | OL1      |       8.99 |
+---------+----------+------------+
8 rows in set (0.09 sec)

-- 这条语句由前面的两条SELECT语句组成，语句中用UNION关键字分隔。
-- UNION指示MySQL执行两条SELECT语句，并把输出组合成单个查询结果集
-- 以下是同样结果,使用where的多条件来实现
select vend_id,prod_id,prod_price from products where prod_price <= 5 or
vend_id in (1001,1002);
+---------+---------+------------+
| vend_id | prod_id | prod_price |
+---------+---------+------------+
|    1001 | ANV01   |       5.99 |
|    1001 | ANV02   |       9.99 |
|    1001 | ANV03   |      14.99 |
|    1003 | FC      |       2.50 |
|    1002 | FU1     |       3.42 |
|    1002 | OL1     |       8.99 |
|    1003 | SLING   |       4.49 |
|    1003 | TNT1    |       2.50 |
+---------+---------+------------+
8 rows in set (0.00 sec)

--在这个简单的例子中，使用UNION可能比使用WHERE子句更为复杂。
--但对于更复杂的过滤条件，或者从多个表（而不是单个表）中检索数据的情形，使用UNION可能会使处理更简单。
```

- 现在思考一个问题,上面的语句分别返回了几条数据?
- 第一条sql返回4行,第二条sql返回5行,那么union返回了几行?

> UNION从查询结果集中自动去除了重复的行（换句话说，它的行为与单条SELECT语句中使用多个WHERE子句条件一样）。

> 这是UNION的默认行为，但是如果需要，可以改变它。如果想返回所有匹配行，可使用UNION ALL而不是UNION

- 对组合查询结果排序

> SELECT语句的输出用ORDER BY子句排序。在用UNION组合查询时，只能使用一条ORDER BY子句，它必须出现在最后一条SELECT语句之后。

> 对于结果集，不存在用一种方式排序一部分，而又用另一种方式排序另一部分的情况，因此不允许使用多条ORDER BY子句。

```
 select vend_id,prod_id,prod_price from products where prod_price <= 5
 union
 select vend_id,prod_id,prod_price from products where vend_id in(1001,1002)
order by  prod_price;
+---------+---------+------------+
| vend_id | prod_id | prod_price |
+---------+---------+------------+
|    1003 | FC      |       2.50 |
|    1003 | TNT1    |       2.50 |
|    1002 | FU1     |       3.42 |
|    1003 | SLING   |       4.49 |
|    1001 | ANV01   |       5.99 |
|    1002 | OL1     |       8.99 |
|    1001 | ANV02   |       9.99 |
|    1001 | ANV03   |      14.99 |
+---------+---------+------------+
8 rows in set (0.00 sec)

--这条UNION在最后一条SELECT语句后使用了ORDER BY子句。
--虽然ORDER BY子句似乎只是最后一条SELECT语句的组成部分，但实际上MySQL将用它来排序所有SELECT语句返回的所有结果。
```