数据库系统课程实验报告

实验名称：	实验四：数据高级查询
实验日期：	2022/4/8
实验地点：	厦门大学德旺图书馆
提交日期：	2022/4/8

学号：	20420192201952
姓名：	庾晓萍
专业年级：	软工 2020 级
学年学期：	2021-2022 学年第二学期

1. 实验目的

- 熟练掌握设计正确的 SQL 查询语句以实现数据高级查询的方法
- 熟练掌握 openGauss 连接查询、子查询和集合查询的语法结构及使用方法

  -（内）连接、（全）外连接、左外连接、右外连接

  -子查询（嵌套查询）

  -不相关子查询与相关子查询

  -EXISTS/NOT EXISTS

  -ANY

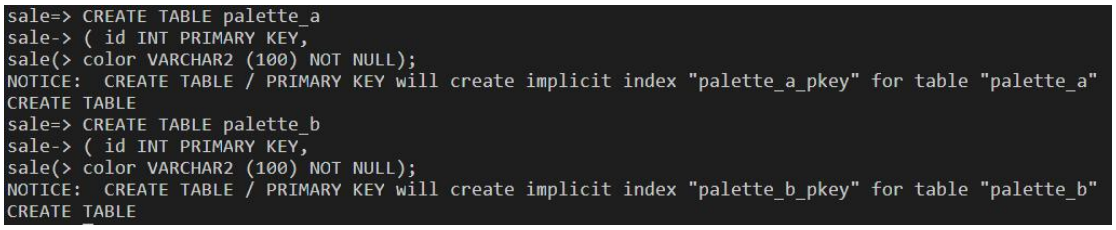
  -ALL

  -集合运算：UNION、INSERSECT、MINUS/EXCEPT
- 理解不相关子查询与相关子查询的不同，掌握构造相应 SQL 语句的方法
- 熟练掌握基于派生表的查询方法
- 建议：对同一查询要求尽量使用不同的查询语句实现。如，所有带 IN 谓词、比较运算符、ANY 或 ALL 谓词的子查询都能用带 EXISTS 谓词的子查询等价替换。

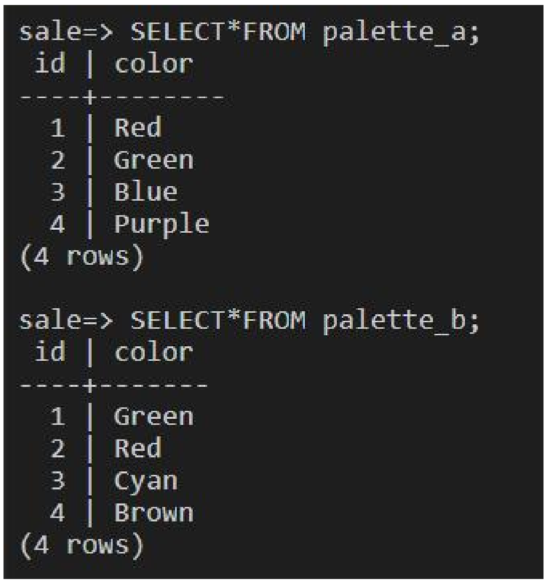
2. 实验内容和步骤

（1）创建两张表 palette_a 和 palette_b(结构相同,但表名不同,color 为颜色)

CREATE TABLE palette_a
(id INT PRIMARY KEY,
color VARCHAR2 (100) NOT NULL);
	CREATE TABLE palette_b
	(id INT PRIMARY KEY,
	color VARCHAR2 (100) NOT NULL);

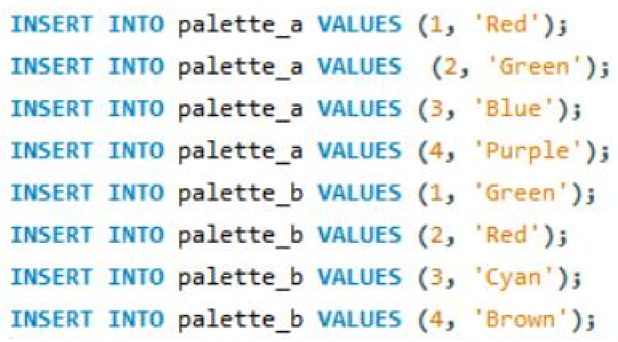


```
sale=> CREATE TABLE palette_a
sale-> ( id INT PRIMARY KEY,
sale(> color VARCHAR2 (100) NOT NULL);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "palette_a_pkey" for table "palette_a"
CREATE TABLE
sale=> CREATE TABLE palette_b
sale-> ( id INT PRIMARY KEY,
sale(> color VARCHAR2 (100) NOT NULL);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "palette_b_pkey" for table "palette_b"
CREATE TABLE
```

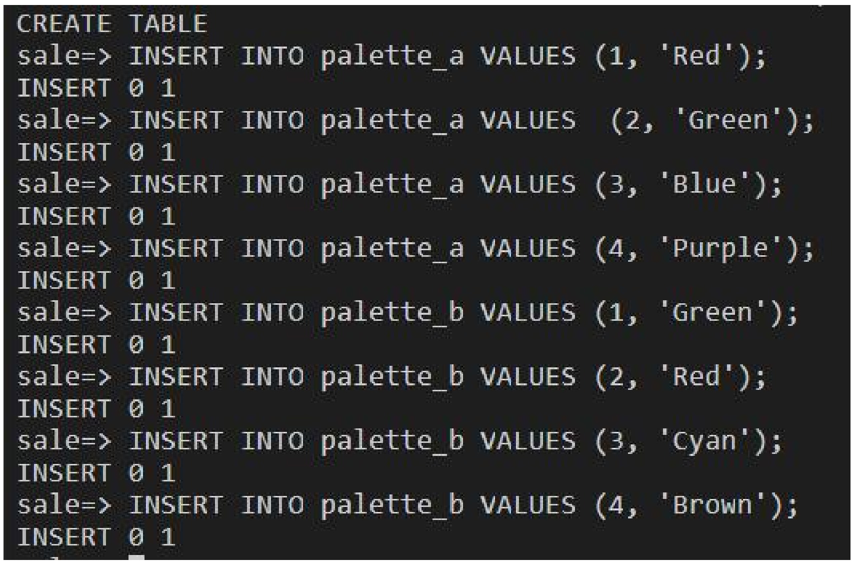


```
sale=> SELECT*FROM palette_a;
 id | color
----+--------
  1 | Red
  2 | Green
  3 | Blue
  4 | Purple
(4 rows)

sale=> SELECT*FROM palette_b;
 id | color
----+-------
  1 | Green
  2 | Red
  3 | Cyan
  4 | Brown
(4 rows)
```

（2）为表 palette_a 添加样例数据：{(1, 'Red'), (2, 'Green'), (3, 'Blue'), (4, 'Purple')}

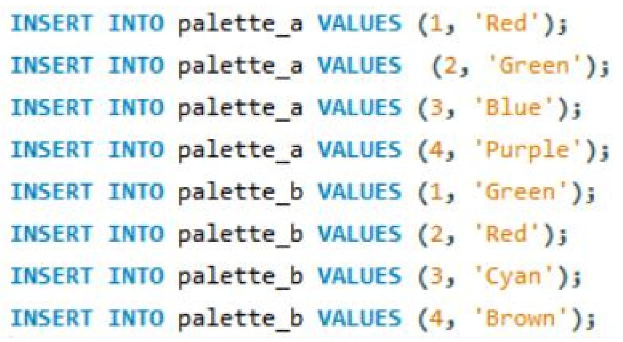


```
INSERT INTO palette_a VALUES (1, 'Red');
INSERT INTO palette_a VALUES  (2, 'Green');
INSERT INTO palette_a VALUES (3, 'Blue');
INSERT INTO palette_a VALUES (4, 'Purple');
INSERT INTO palette_b VALUES (1, 'Green');
INSERT INTO palette_b VALUES (2, 'Red');
INSERT INTO palette_b VALUES (3, 'Cyan');
INSERT INTO palette_b VALUES (4, 'Brown');
```

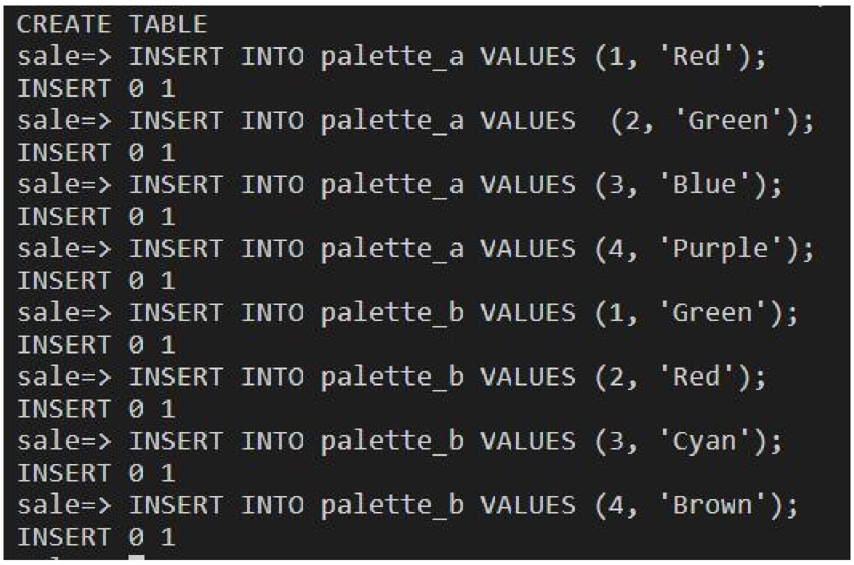


```
CREATE TABLE
sale=> INSERT INTO palette_a VALUES (1, 'Red');
INSERT 0 1
sale=> INSERT INTO palette_a VALUES  (2, 'Green');
INSERT 0 1
sale=> INSERT INTO palette_a VALUES (3, 'Blue');
INSERT 0 1
sale=> INSERT INTO palette_a VALUES (4, 'Purple');
INSERT 0 1
sale=> INSERT INTO palette_b VALUES (1, 'Green');
INSERT 0 1
sale=> INSERT INTO palette_b VALUES (2, 'Red');
INSERT 0 1
sale=> INSERT INTO palette_b VALUES (3, 'Cyan');
INSERT 0 1
sale=> INSERT INTO palette_b VALUES (4, 'Brown');
INSERT 0 1
```

（3）为表 palette_b 添加样例数据：{(1, 'Green'), (2, 'Red'), (3, 'Cyan'), (4, 'Brown')}

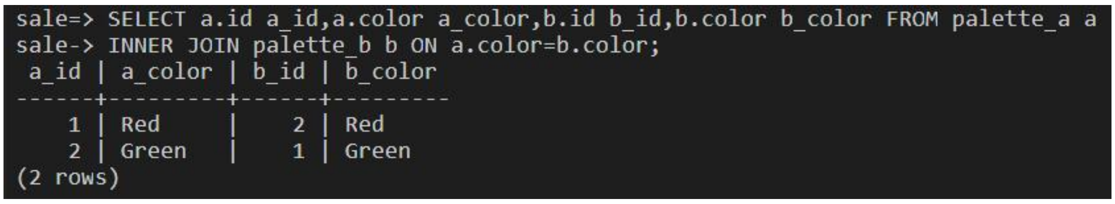


```
INSERT INTO palette_a VALUES (1, 'Red');
INSERT INTO palette_a VALUES  (2, 'Green');
INSERT INTO palette_a VALUES (3, 'Blue');
INSERT INTO palette_a VALUES (4, 'Purple');
INSERT INTO palette_b VALUES (1, 'Green');
INSERT INTO palette_b VALUES (2, 'Red');
INSERT INTO palette_b VALUES (3, 'Cyan');
INSERT INTO palette_b VALUES (4, 'Brown');
```



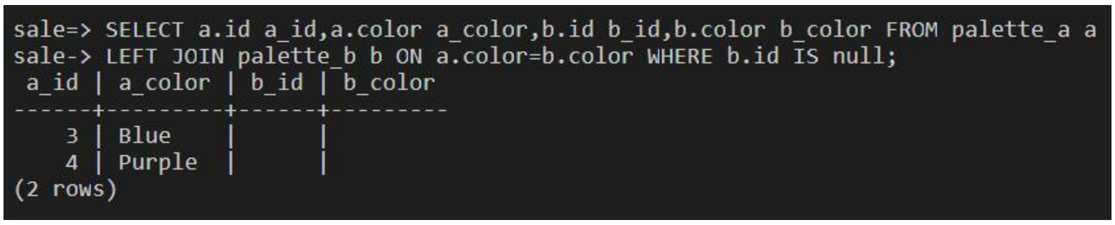
```
CREATE TABLE
sale=> INSERT INTO palette_a VALUES (1, 'Red');
INSERT 0 1
sale=> INSERT INTO palette_a VALUES  (2, 'Green');
INSERT 0 1
sale=> INSERT INTO palette_a VALUES (3, 'Blue');
INSERT 0 1
sale=> INSERT INTO palette_a VALUES (4, 'Purple');
INSERT 0 1
sale=> INSERT INTO palette_b VALUES (1, 'Green');
INSERT 0 1
sale=> INSERT INTO palette_b VALUES (2, 'Red');
INSERT 0 1
sale=> INSERT INTO palette_b VALUES (3, 'Cyan');
INSERT 0 1
sale=> INSERT INTO palette_b VALUES (4, 'Brown');
INSERT 0 1
```

（4）查询两张表中相同颜色的所有信息



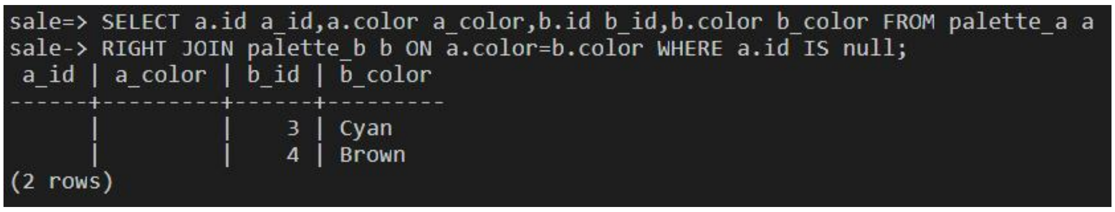
```
sale=> SELECT a.id a_id,a.color a_color,b.id b_id,b.color b_color FROM palette_a a
sale-> INNER JOIN palette_b b ON a.color=b.color;
 a_id | a_color | b_id | b_color
------+---------+------+---------
    1 | Red     |    2 | Red
    2 | Green   |    1 | Green
(2 rows)
```

（5）查询 palette_a 表中颜色不出现在 palette_b 表中的两张表的 id 和颜色（用左外连接）



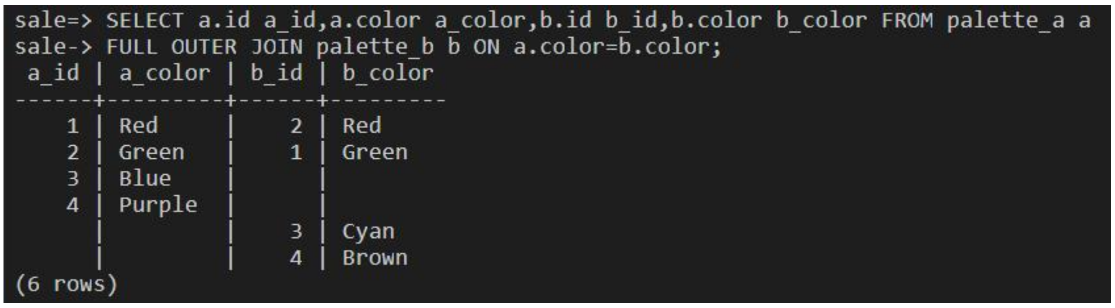
```
sale=> SELECT a.id a_id,a.color a_color,b.id b_id,b.color b_color FROM palette_a a
sale-> LEFT JOIN palette_b b ON a.color=b.color WHERE b.id IS null;
 a_id | a_color | b_id | b_color
------+---------+------+---------
    3 | Blue    |      |
    4 | Purple  |      |
(2 rows)
```

（6）查询 palette_b 表中颜色不出现在 palette_a 表中的两张表的 id 和颜色（用右外连接）



```
sale=> SELECT a.id a_id,a.color a_color,b.id b_id,b.color b_color FROM palette_a a
sale-> RIGHT JOIN palette_b b ON a.color=b.color WHERE a.id IS null;
 a_id | a_color | b_id | b_color
------+---------+------+---------
      |         |    3 | Cyan
      |         |    4 | Brown
(2 rows)
```

（7）查询（5）或（6）两种情况的信息（用（全）外连接）

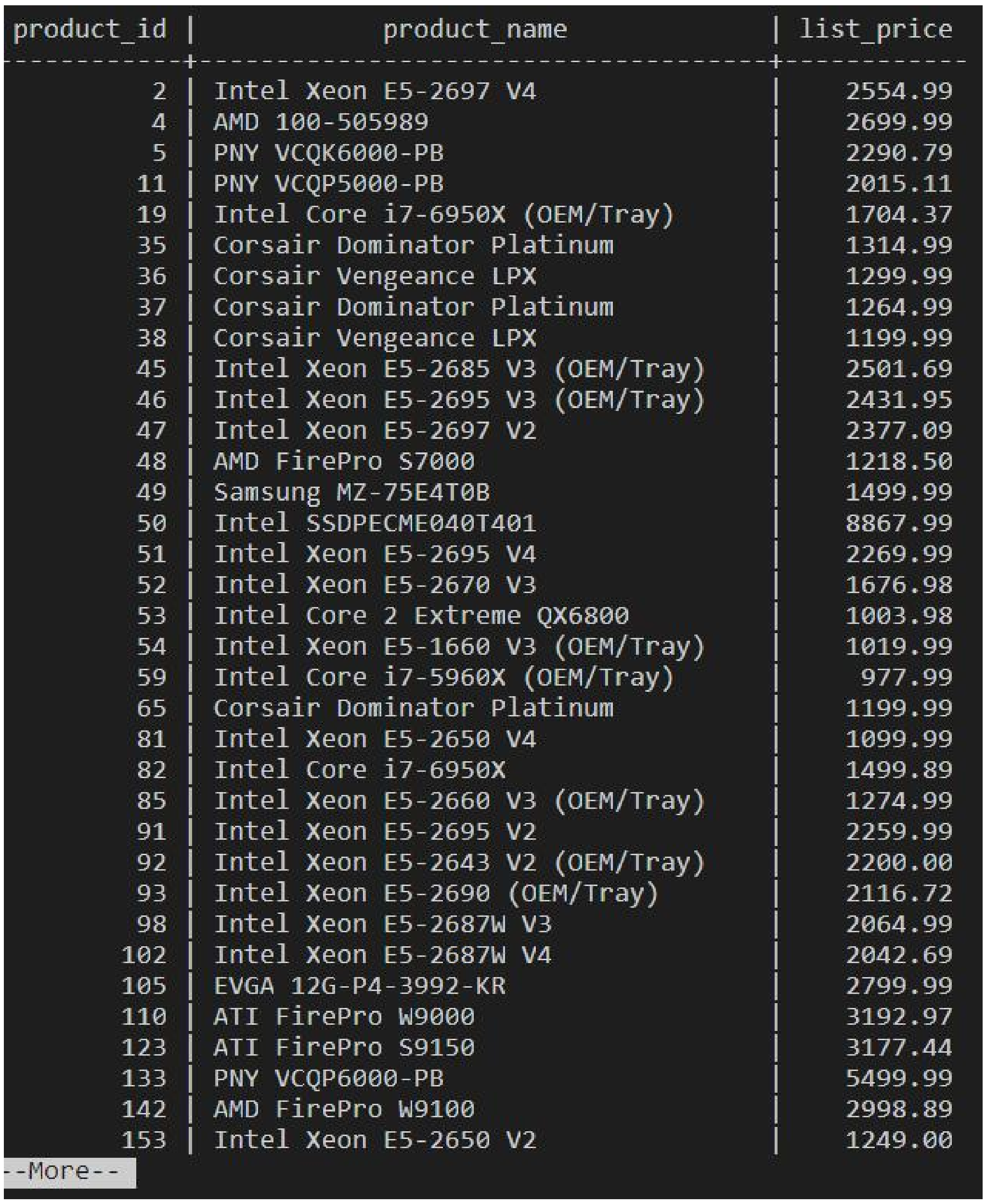


```
sale=> SELECT a.id a_id,a.color a_color,b.id b_id,b.color b_color FROM palette_a a
sale-> FULL OUTER JOIN palette_b b ON a.color=b.color;
 a_id | a_color | b_id | b_color
------+---------+------+---------
    1 | Red     |    2 | Red
    2 | Green   |    1 | Green
    3 | Blue    |      |
    4 | Purple  |      |
      |         |    3 | Cyan
      |         |    4 | Brown
(6 rows)
```

（8）查询产品表 products 中的 product_id, product_name, list_price 信息，要求产品定价 list_price 大于其平均定价 list_price。

思路：聚合函数不允许直接在 WHERE 语句中，故首先子查询返回所有产品的平均标价。其次外部查询获取标价大于子查询返回的平均标价的产品。

```
聚合函数不允许直接在WHERE语句中*/
SELECT product_id, product_name, list_price FROM products WHERE  list_price>
(SELECT  AVG(list_price) FROM  products) ORDER BY product_id;
```

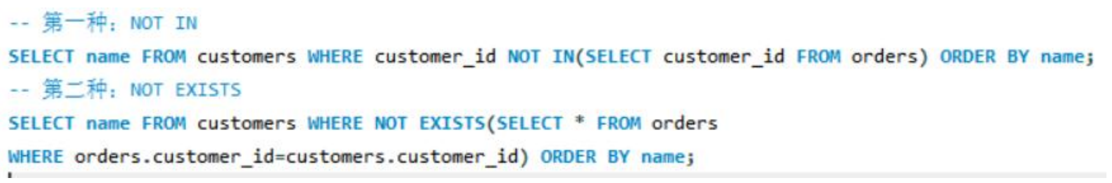


```
product_id |              product_name              | list_price
-----------+----------------------------------------+------------
         2 | Intel Xeon E5-2697 V4                  |    2554.99
         4 | AMD 100-505989                         |    2699.99
         5 | PNY VCQK6000-PB                        |    2290.79
        11 | PNY VCQP5000-PB                        |    2015.11
        19 | Intel Core i7-6950X (OEM/Tray)         |    1704.37
        35 | Corsair Dominator Platinum             |    1314.99
        36 | Corsair Vengeance LPX                  |    1299.99
        37 | Corsair Dominator Platinum             |    1264.99
        38 | Corsair Vengeance LPX                  |    1199.99
        45 | Intel Xeon E5-2685 V3 (OEM/Tray)       |    2501.69
        46 | Intel Xeon E5-2695 V3 (OEM/Tray)       |    2431.95
        47 | Intel Xeon E5-2697 V2                  |    2377.09
        48 | AMD FirePro S7000                      |    1218.50
        49 | Samsung MZ-75E4T0B                     |    1499.99
        50 | Intel SSDPECME040T401                  |    8867.99
        51 | Intel Xeon E5-2695 V4                  |    2269.99
        52 | Intel Xeon E5-2670 V3                  |    1676.98
        53 | Intel Core 2 Extreme QX6800            |    1003.98
        54 | Intel Xeon E5-1660 V3 (OEM/Tray)       |    1019.99
        59 | Intel Core i7-5960X (OEM/Tray)         |     977.99
        65 | Corsair Dominator Platinum             |    1199.99
        81 | Intel Xeon E5-2650 V4                  |    1099.99
        82 | Intel Core i7-6950X                    |    1499.89
        85 | Intel Xeon E5-2660 V3 (OEM/Tray)       |    1274.99
        91 | Intel Xeon E5-2695 V2                  |    2259.99
        92 | Intel Xeon E5-2643 V2 (OEM/Tray)       |    2200.00
        93 | Intel Xeon E5-2690 (OEM/Tray)          |    2116.72
        98 | Intel Xeon E5-2687W V3                 |    2064.99
       102 | Intel Xeon E5-2687W V4                 |    2042.69
       105 | EVGA 12G-P4-3992-KR                    |    2799.99
       110 | ATI FirePro W9000                      |    3192.97
       123 | ATI FirePro S9150                      |    3177.44
       133 | PNY VCQP6000-PB                        |    5499.99
       142 | AMD FirePro W9100                      |    2998.89
       153 | Intel Xeon E5-2650 V2                  |    1249.00
--More--
```

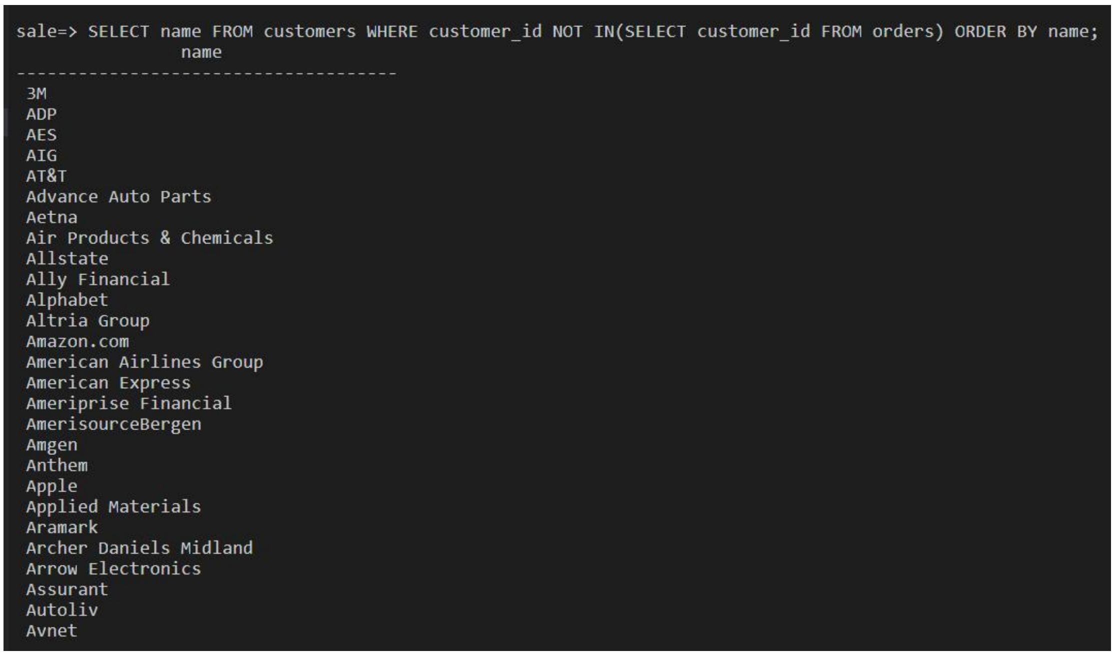
（9）查询没有一个订单的顾客姓名（实现要求：NOT IN（必须）+其它查询方法（如果找到））

第一种可以使用 NOT IN，SELECT name FROM customers WHERE customer_id NOT IN(SELECT customer_id FROM orders) ORDER BY name; 第二种可以使用 NOT EXISTS, SELECT name FROM customers WHERE NOT EXISTS(SELECT * FROM orders
WHERE orders.customer_id=customers.customer_id) ORDER BY name;

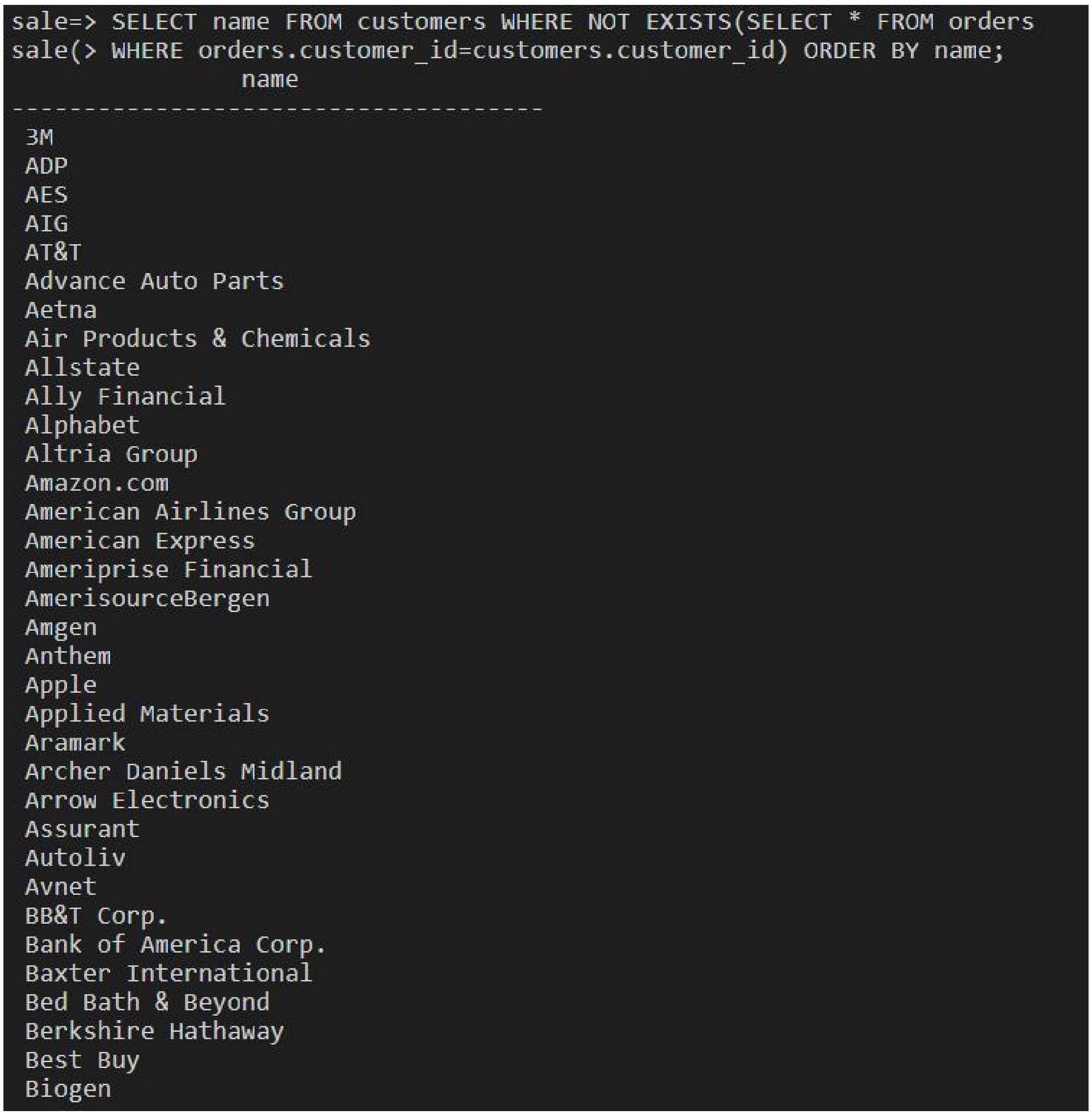
输出的结果相同。



```
-- 第一种：NOT IN
SELECT name FROM customers WHERE customer_id NOT IN(SELECT customer_id FROM orders) ORDER BY name;
-- 第二种：NOT EXISTS
SELECT name FROM customers WHERE NOT EXISTS(SELECT * FROM orders
WHERE orders.customer_id=customers.customer_id) ORDER BY name;
```



```
sale=> SELECT name FROM customers WHERE customer_id NOT IN(SELECT customer_id FROM orders) ORDER BY name;
                name
-------------------------------------
 3M
 ADP
 AES
 AIG
 AT&T
 Advance Auto Parts
 Aetna
 Air Products & Chemicals
 Allstate
 Ally Financial
 Alphabet
 Altria Group
 Amazon.com
 American Airlines Group
 American Express
 Ameriprise Financial
 AmerisourceBergen
 Amgen
 Anthem
 Apple
 Applied Materials
 Aramark
 Archer Daniels Midland
 Arrow Electronics
 Assurant
 Autoliv
 Avnet
```



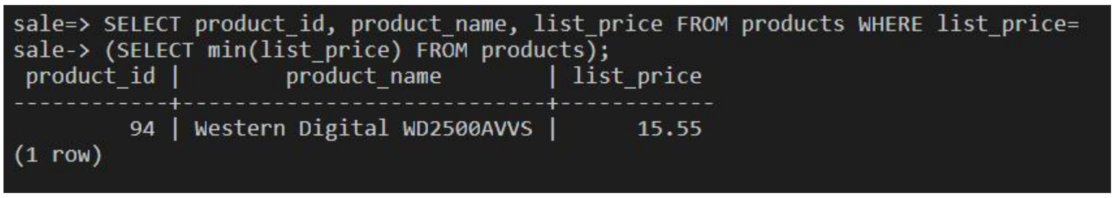
```
sale=> SELECT name FROM customers WHERE NOT EXISTS(SELECT * FROM orders
sale(> WHERE orders.customer_id=customers.customer_id) ORDER BY name;
                name
-------------------------------------
 3M
 ADP
 AES
 AIG
 AT&T
 Advance Auto Parts
 Aetna
 Air Products & Chemicals
 Allstate
 Ally Financial
 Alphabet
 Altria Group
 Amazon.com
 American Airlines Group
 American Express
 Ameriprise Financial
 AmerisourceBergen
 Amgen
 Anthem
 Apple
 Applied Materials
 Aramark
 Archer Daniels Midland
 Arrow Electronics
 Assurant
 Autoliv
 Avnet
 BB&T Corp.
 Bank of America Corp.
 Baxter International
 Bed Bath & Beyond
 Berkshire Hathaway
 Best Buy
 Biogen
```

(10) 查询产品表 products 中最便宜产品的 product_id, product_name, list_price。

思路：首先独立执行子查询。其次 Oracle 只对子查询求值一次。子查询返回结果集后，外部查询使用它们。换句话说，外部查询依赖于子查询。但是，子查询是独立的，不依赖于外部查询的值。

相关子查询是使用来自外部查询的值的子查询。因此，使用相关子查询的查询可能会很慢。

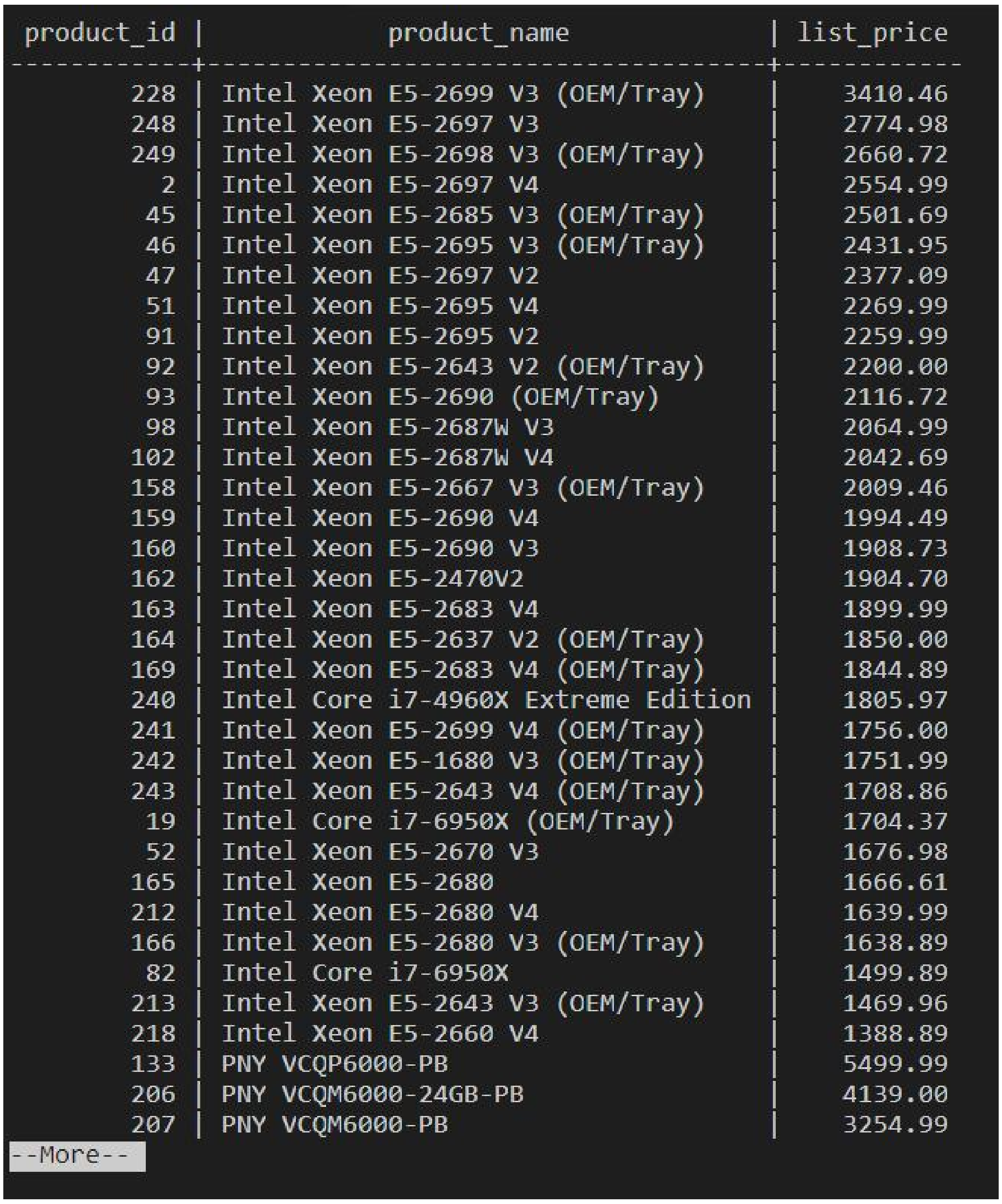
```
SELECT product_id, product_name, list_price FROM products WHERE list_price=
(SELECT min(list_price) FROM products);
```



```
sale=> SELECT product_id, product_name, list_price FROM products WHERE list_price=
sale-> (SELECT min(list_price) FROM products);
 product_id |         product_name         | list_price
------------+-----------------------------+------------
         94 | Western Digital WD2500AVVS |      15.55
(1 row)
```

（11）查询产品表 products 中产品的 product_id, product_name, list_price，要求产品定价 list_price 大于其同类产品（可由 category_id 表达）的平均定价。实现要求：相关子查询（必须）+基于派生表的查询（如果找到）

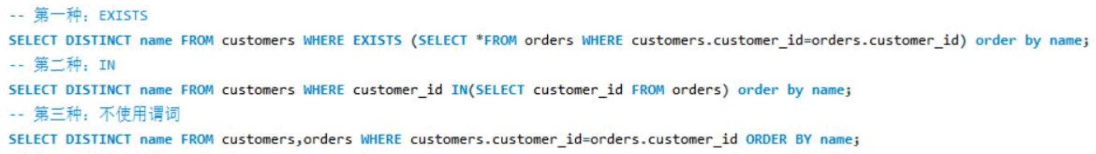
```
SELECT product_id, product_name, list_price FROM products a WHERE list_price>
(SELECT avg(list_price) FROM products b WHERE a.category_id=b.category_id)
```



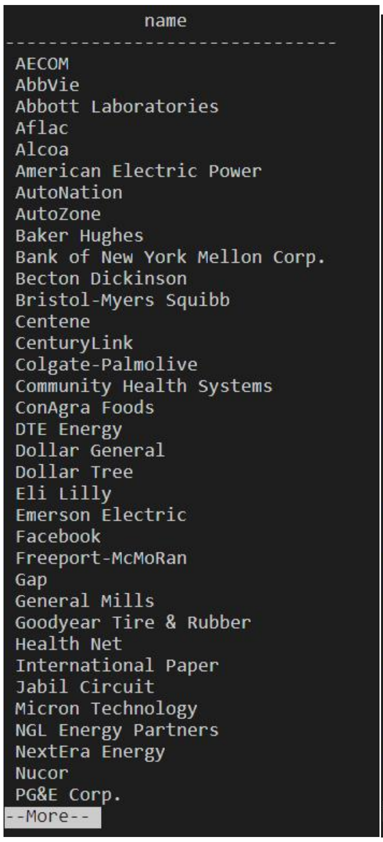
```
 product_id |              product_name              | list_price
------------+----------------------------------------+------------
        228 | Intel Xeon E5-2699 V3 (OEM/Tray)       |    3410.46
        248 | Intel Xeon E5-2697 V3                  |    2774.98
        249 | Intel Xeon E5-2698 V3 (OEM/Tray)       |    2660.72
          2 | Intel Xeon E5-2697 V4                  |    2554.99
         45 | Intel Xeon E5-2685 V3 (OEM/Tray)       |    2501.69
         46 | Intel Xeon E5-2695 V3 (OEM/Tray)       |    2431.95
         47 | Intel Xeon E5-2697 V2                  |    2377.09
         51 | Intel Xeon E5-2695 V4                  |    2269.99
         91 | Intel Xeon E5-2695 V2                  |    2259.99
         92 | Intel Xeon E5-2643 V2 (OEM/Tray)       |    2200.00
         93 | Intel Xeon E5-2690 (OEM/Tray)          |    2116.72
         98 | Intel Xeon E5-2687W V3                 |    2064.99
        102 | Intel Xeon E5-2687W V4                 |    2042.69
        158 | Intel Xeon E5-2667 V3 (OEM/Tray)       |    2009.46
        159 | Intel Xeon E5-2690 V4                  |    1994.49
        160 | Intel Xeon E5-2690 V3                  |    1908.73
        162 | Intel Xeon E5-2470V2                   |    1904.70
        163 | Intel Xeon E5-2683 V4                  |    1899.99
        164 | Intel Xeon E5-2637 V2 (OEM/Tray)       |    1850.00
        169 | Intel Xeon E5-2683 V4 (OEM/Tray)       |    1844.89
        240 | Intel Core i7-4960X Extreme Edition    |    1805.97
        241 | Intel Xeon E5-2699 V4 (OEM/Tray)       |    1756.00
        242 | Intel Xeon E5-1680 V3 (OEM/Tray)       |    1751.99
        243 | Intel Xeon E5-2643 V4 (OEM/Tray)       |    1708.86
         19 | Intel Core i7-6950X (OEM/Tray)         |    1704.37
         52 | Intel Xeon E5-2670 V3                  |    1676.98
        165 | Intel Xeon E5-2680                     |    1666.61
        212 | Intel Xeon E5-2680 V4                  |    1639.99
        166 | Intel Xeon E5-2680 V3 (OEM/Tray)       |    1638.89
         82 | Intel Core i7-6950X                    |    1499.89
        213 | Intel Xeon E5-2643 V3 (OEM/Tray)       |    1469.96
        218 | Intel Xeon E5-2660 V4                  |    1388.89
        133 | PNY VCQP6000-PB                        |    5499.99
        206 | PNY VCQM6000-24GB-PB                   |    4139.00
        207 | PNY VCQM6000-PB                        |    3254.99
--More--
```

（12）查询有订单 order 的所有顾客 customer 姓名（查询涉及 customers 表和 orders 表）实现要求：使用 EXISTS（必须）+其它查询方法（如果找到）

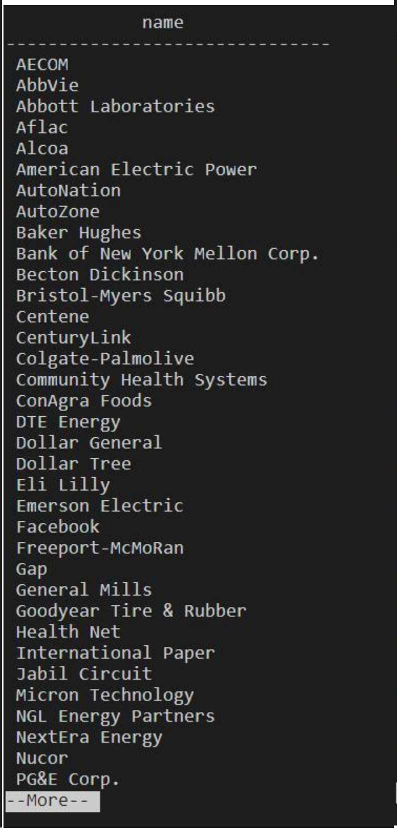
可以使用 EXISTS、IN 或者不用谓词，结果输出相同。



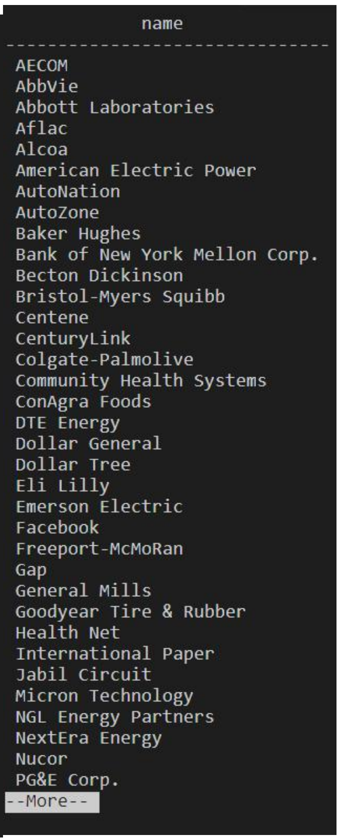
```sql
-- 第一种：EXISTS
SELECT DISTINCT name FROM customers WHERE EXISTS (SELECT *FROM orders WHERE customers.customer_id=orders.customer_id) order by name;
-- 第二种：IN
SELECT DISTINCT name FROM customers WHERE customer_id IN(SELECT customer_id FROM orders) order by name;
-- 第三种：不使用谓词
SELECT DISTINCT name FROM customers,orders WHERE customers.customer_id=orders.customer_id ORDER BY name;
```



```
              name
-------------------------------
 AECOM
 AbbVie
 Abbott Laboratories
 Aflac
 Alcoa
 American Electric Power
 AutoNation
 AutoZone
 Baker Hughes
 Bank of New York Mellon Corp.
 Becton Dickinson
 Bristol-Myers Squibb
 Centene
 CenturyLink
 Colgate-Palmolive
 Community Health Systems
 ConAgra Foods
 DTE Energy
 Dollar General
 Dollar Tree
 Eli Lilly
 Emerson Electric
 Facebook
 Freeport-McMoRan
 Gap
 General Mills
 Goodyear Tire & Rubber
 Health Net
 International Paper
 Jabil Circuit
 Micron Technology
 NGL Energy Partners
 NextEra Energy
 Nucor
 PG&E Corp.
--More--
```



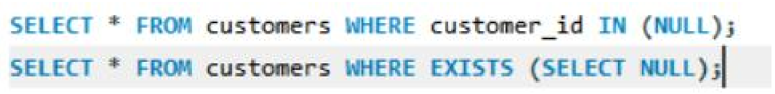
```
              name
-------------------------------
 AECOM
 AbbVie
 Abbott Laboratories
 Aflac
 Alcoa
 American Electric Power
 AutoNation
 AutoZone
 Baker Hughes
 Bank of New York Mellon Corp.
 Becton Dickinson
 Bristol-Myers Squibb
 Centene
 CenturyLink
 Colgate-Palmolive
 Community Health Systems
 ConAgra Foods
 DTE Energy
 Dollar General
 Dollar Tree
 Eli Lilly
 Emerson Electric
 Facebook
 Freeport-McMoRan
 Gap
 General Mills
 Goodyear Tire & Rubber
 Health Net
 International Paper
 Jabil Circuit
 Micron Technology
 NGL Energy Partners
 NextEra Energy
 Nucor
 PG&E Corp.
--More--
```



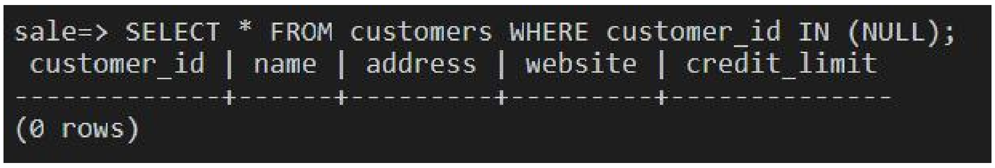
```
              name
-------------------------------
 AECOM
 AbbVie
 Abbott Laboratories
 Aflac
 Alcoa
 American Electric Power
 AutoNation
 AutoZone
 Baker Hughes
 Bank of New York Mellon Corp.
 Becton Dickinson
 Bristol-Myers Squibb
 Centene
 CenturyLink
 Colgate-Palmolive
 Community Health Systems
 ConAgra Foods
 DTE Energy
 Dollar General
 Dollar Tree
 Eli Lilly
 Emerson Electric
 Facebook
 Freeport-McMoRan
 Gap
 General Mills
 Goodyear Tire & Rubber
 Health Net
 International Paper
 Jabil Circuit
 Micron Technology
 NGL Energy Partners
 NextEra Energy
 Nucor
 PG&E Corp.
--More--
```

（13）执行以下两条语句，观察有何不同，能否得出某些初步结论?

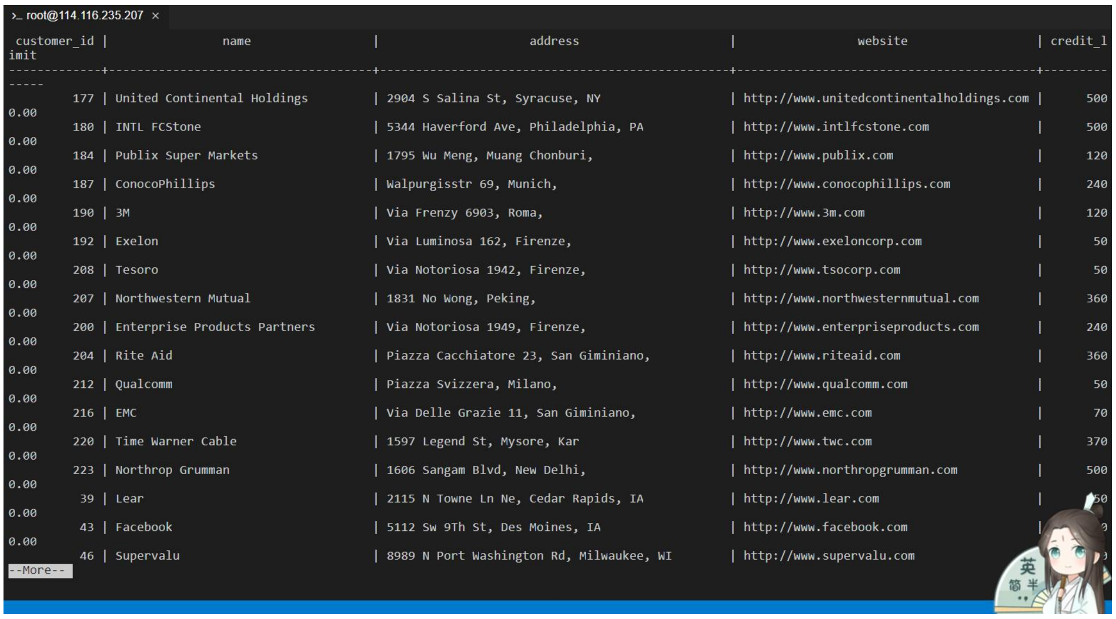
结论：一旦子查询返回第一行，EXISTS 运算符就会停止扫描行，因为它可以确定结果，而 IN 运算符必须扫描子查询返回的所有行才能得出结果。此外，IN 子句不能将任何内容与 NULL 值进行比较，但该 EXISTS 子句可以将所有内容与 NULL 值进行比较。例如，第一个语句不返回任何行，而第二个语句返回 customers 表中的所有行。通常，当子查询的结果集很大时，EXISTS 运算符比 IN 运算符快。相比之下，当子查询的结果集较小时，IN 运算符比 EXISTS 运算符快。（注释：openguass 不需要 FROM dual，可以将 FROM dual 直接删去）



```sql
SELECT * FROM customers WHERE customer_id IN (NULL);
SELECT * FROM customers WHERE EXISTS (SELECT NULL);
```



```
sale=> SELECT * FROM customers WHERE customer_id IN (NULL);
 customer_id | name | address | website | credit_limit
-------------+------+---------+---------+--------------
(0 rows)
```

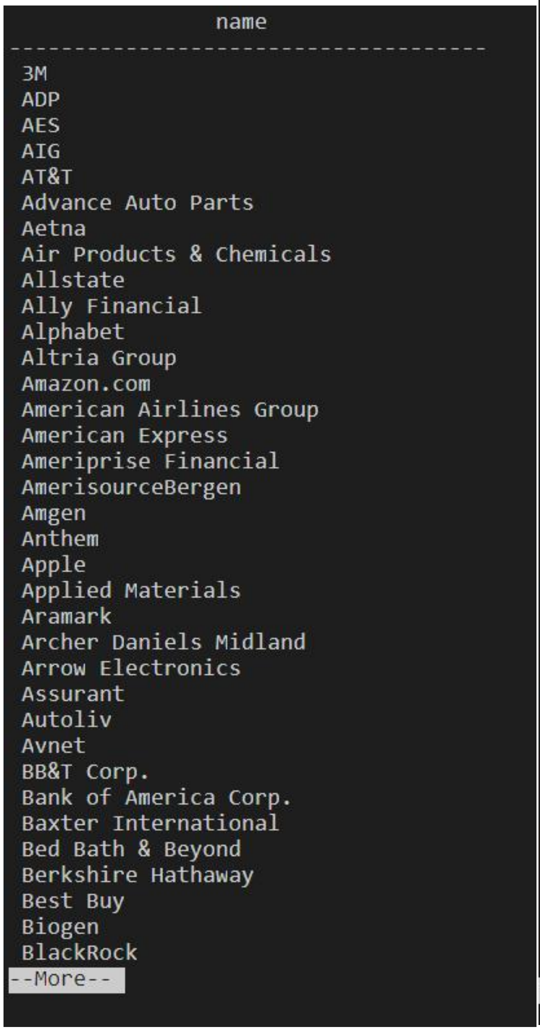


```
>_ root@114.116.235.207 ×
 customer_id |                 name                |                  address                  |                  website                  | credit_l
imit
-------------+-------------------------------------+-------------------------------------------+-------------------------------------------+---------
-----
         177 | United Continental Holdings         | 2904 S Salina St, Syracuse, NY            | http://www.unitedcontinentalholdings.com |      500
0.00
         180 | INTL FCStone                        | 5344 Haverford Ave, Philadelphia, PA      | http://www.intlfcstone.com                |      500
0.00
         184 | Publix Super Markets                | 1795 Wu Meng, Muang Chonburi,             | http://www.publix.com                     |      120
0.00
         187 | ConocoPhillips                      | Walpurgisstr 69, Munich,                  | http://www.conocophillips.com             |      240
0.00
         190 | 3M                                  | Via Frenzy 6903, Roma,                    | http://www.3m.com                         |      120
0.00
         192 | Exelon                              | Via Luminosa 162, Firenze,                | http://www.exeloncorp.com                 |       50
0.00
         208 | Tesoro                              | Via Notoriosa 1942, Firenze,              | http://www.tsocorp.com                    |       50
0.00
         207 | Northwestern Mutual                 | 1831 No Wong, Peking,                     | http://www.northwesternmutual.com         |      360
0.00
         200 | Enterprise Products Partners        | Via Notoriosa 1949, Firenze,              | http://www.enterpriseproducts.com         |      240
0.00
         204 | Rite Aid                            | Piazza Cacchiatore 23, San Giminiano,     | http://www.riteaid.com                    |      360
0.00
         212 | Qualcomm                            | Piazza Svizzera, Milano,                  | http://www.qualcomm.com                   |       50
0.00
         216 | EMC                                 | Via Delle Grazie 11, San Giminiano,       | http://www.emc.com                        |       70
0.00
         220 | Time Warner Cable                   | 1597 Legend St, Mysore, Kar               | http://www.twc.com                        |      370
0.00
         223 | Northrop Grumman                    | 1606 Sangam Blvd, New Delhi,              | http://www.northropgrumman.com            |      500
0.00
          39 | Lear                                | 2115 N Towne Ln Ne, Cedar Rapids, IA      | http://www.lear.com                       |       50
0.00
          43 | Facebook                            | 5112 Sw 9Th St, Des Moines, IA            | http://www.facebook.com                   | [illegible]
0.00
          46 | Supervalu                           | 8989 N Port Washington Rd, Milwaukee, WI  | http://www.supervalu.com                  | [illegible]
--More--
```

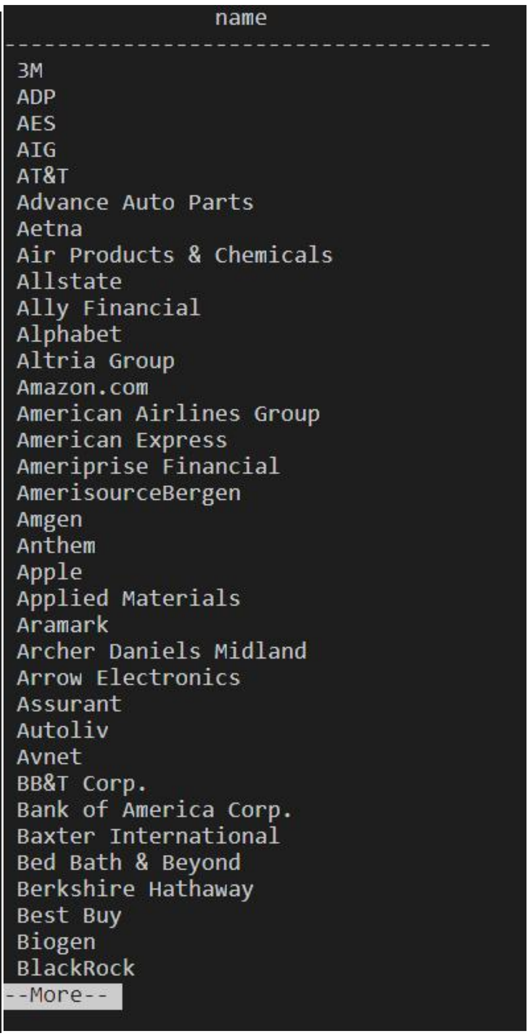
（14）找出所有没有订单的顾客姓名（查询涉及 customers 表和 orders 表）实现要求：使用 NOT EXISTS（必须）+其它查询方法（如果找到）

思路：可以使用 NOT IN 和 NOT EXISTS 两种方法。

```
-- 第一种 NOT EXISTS
SELECT DISTINCT name FROM customers WHERE NOT EXISTS
(SELECT *FROM orders WHERE customers.customer_id=orders.customer_id) order by name;
-- 第二种 NOT IN
SELECT DISTINCT name FROM customers WHERE customers.customer_id NOT IN  (SELECT customer_id FROM orders) order by name;
```



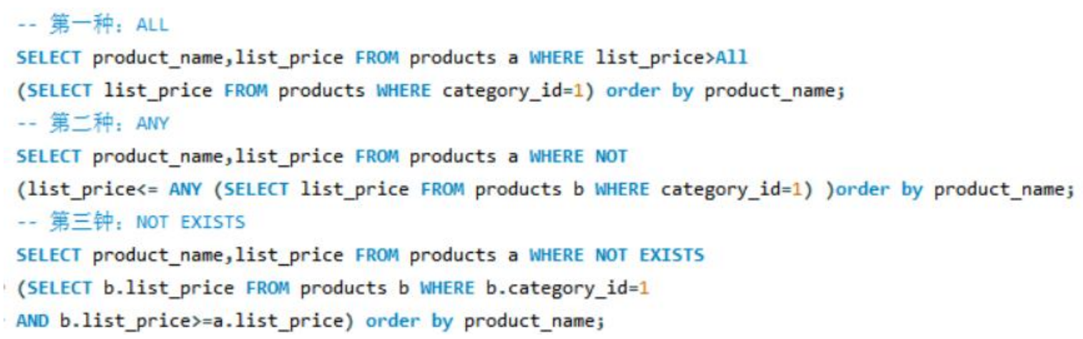
```
                name
------------------------------------
 3M
 ADP
 AES
 AIG
 AT&T
 Advance Auto Parts
 Aetna
 Air Products & Chemicals
 Allstate
 Ally Financial
 Alphabet
 Altria Group
 Amazon.com
 American Airlines Group
 American Express
 Ameriprise Financial
 AmerisourceBergen
 Amgen
 Anthem
 Apple
 Applied Materials
 Aramark
 Archer Daniels Midland
 Arrow Electronics
 Assurant
 Autoliv
 Avnet
 BB&T Corp.
 Bank of America Corp.
 Baxter International
 Bed Bath & Beyond
 Berkshire Hathaway
 Best Buy
 Biogen
 BlackRock
--More--
```



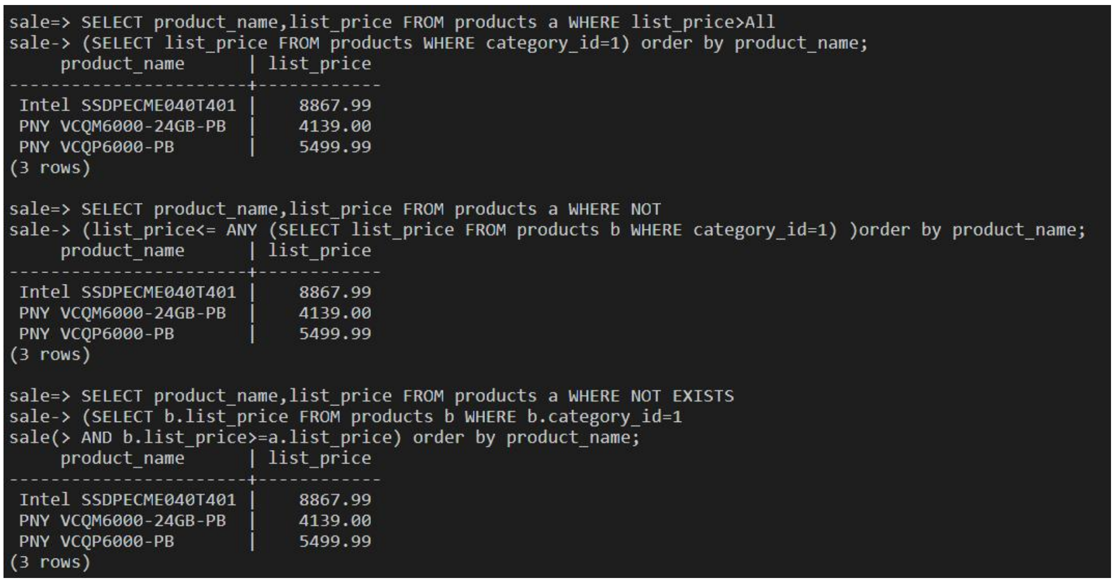
```
                name
------------------------------------
3M
ADP
AES
AIG
AT&T
Advance Auto Parts
Aetna
Air Products & Chemicals
Allstate
Ally Financial
Alphabet
Altria Group
Amazon.com
American Airlines Group
American Express
Ameriprise Financial
AmerisourceBergen
Amgen
Anthem
Apple
Applied Materials
Aramark
Archer Daniels Midland
Arrow Electronics
Assurant
Autoliv
Avnet
BB&T Corp.
Bank of America Corp.
Baxter International
Bed Bath & Beyond
Berkshire Hathaway
Best Buy
Biogen
BlackRock
--More--
```

（15）查询产品表 products 中的产品名 product_name 和定价 list_price，要求其定价高于产品种类 1 中的任何产品定价。实现要求：ANY（必须）+其它查询方法（如果找到）

思路：可以使用三种方法：ALL，ANY，NOT EXISTS。

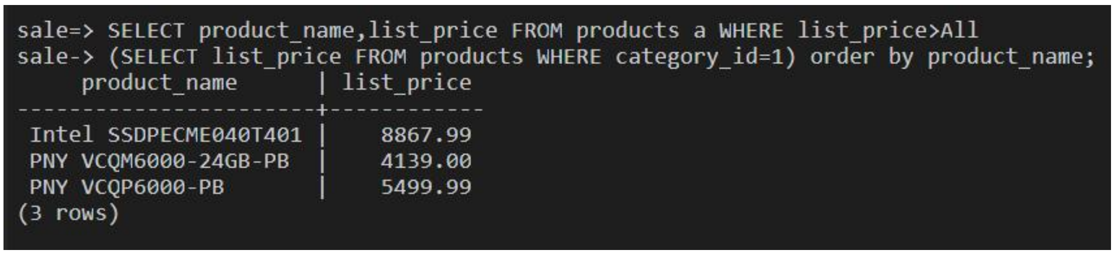


```sql
-- 第一种：ALL
SELECT product_name,list_price FROM products a WHERE list_price>All
(SELECT list_price FROM products WHERE category_id=1) order by product_name;
-- 第二种：ANY
SELECT product_name,list_price FROM products a WHERE NOT
(list_price<= ANY (SELECT list_price FROM products b WHERE category_id=1) )order by product_name;
-- 第三钟：NOT EXISTS
SELECT product_name,list_price FROM products a WHERE NOT EXISTS
(SELECT b.list_price FROM products b WHERE b.category_id=1
AND b.list_price>=a.list_price) order by product_name;
```


```
sale=> SELECT product_name,list_price FROM products a WHERE list_price>All
sale-> (SELECT list_price FROM products WHERE category_id=1) order by product_name;
     product_name       | list_price
------------------------+------------
 Intel SSDPECME040T401 |    8867.99
 PNY VCQM6000-24GB-PB  |    4139.00
 PNY VCQP6000-PB       |    5499.99
(3 rows)

sale=> SELECT product_name,list_price FROM products a WHERE NOT
sale-> (list_price<= ANY (SELECT list_price FROM products b WHERE category_id=1) )order by product_name;
     product_name       | list_price
------------------------+------------
 Intel SSDPECME040T401 |    8867.99
 PNY VCQM6000-24GB-PB  |    4139.00
 PNY VCQP6000-PB       |    5499.99
(3 rows)

sale=> SELECT product_name,list_price FROM products a WHERE NOT EXISTS
sale-> (SELECT b.list_price FROM products b WHERE b.category_id=1
sale(> AND b.list_price>=a.list_price) order by product_name;
     product_name       | list_price
------------------------+------------
 Intel SSDPECME040T401 |    8867.99
 PNY VCQM6000-24GB-PB  |    4139.00
 PNY VCQP6000-PB       |    5499.99
(3 rows)
```

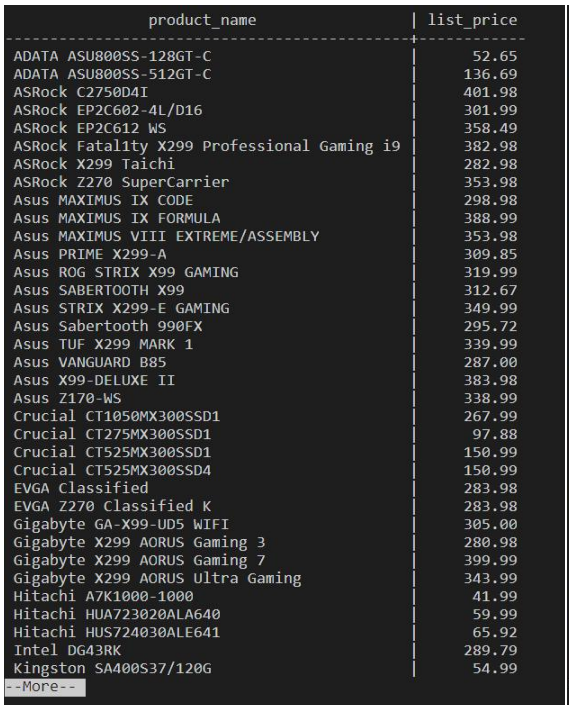
（16）查询产品表 products 中的产品名 product_name 和定价 list_price，要求其定价高于产品种类 1 中的所有定价。

```
SELECT product_name,list_price FROM products a WHERE list_price>All
(SELECT list_price FROM products WHERE category_id=1) order by product_name;
```

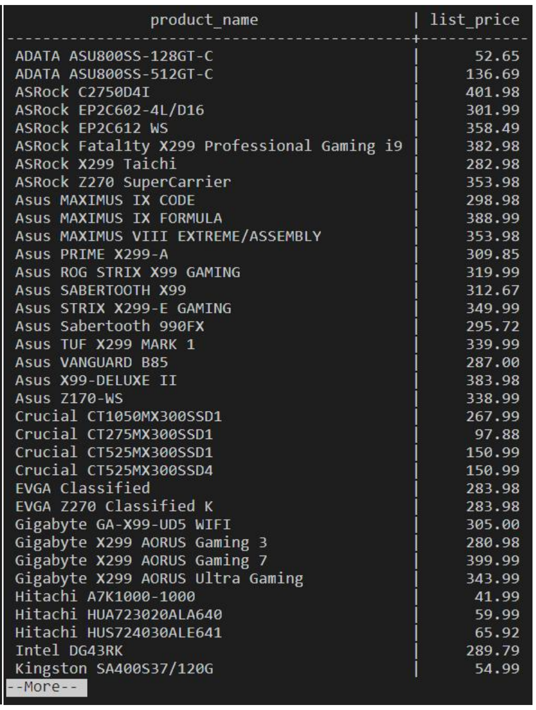

```
sale=> SELECT product_name,list_price FROM products a WHERE list_price>All
sale-> (SELECT list_price FROM products WHERE category_id=1) order by product_name;
     product_name       | list_price
------------------------+------------
 Intel SSDPECME040T401 |    8867.99
 PNY VCQM6000-24GB-PB  |    4139.00
 PNY VCQP6000-PB       |    5499.99
(3 rows)
```

（17）查询产品表 products 中的产品名 product_name 和定价 list_price，要求其定价低于产品种类的所有平均定价。实现要求：ALL（必须）+其它查询方法（如果找到）

```
-- 第一种：ALL
SELECT product_name,list_price FROM products WHERE list_price<
ALL(SELECT AVG(list_price)FROM products GROUP BY category_id) order by product_name;
-- 第二种：ANY
SELECT product_name,list_price FROM products WHERE NOT
(list_price>=ANY(SELECT AVG(list_price)FROM products GROUP BY category_id)) order by product_name;
```

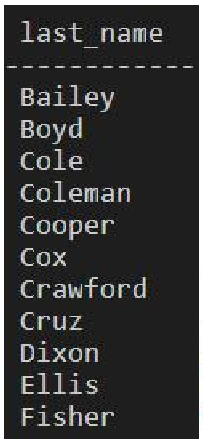


```
                 product_name                  | list_price
-----------------------------------------------+------------
 ADATA ASU800SS-128GT-C                        |      52.65
 ADATA ASU800SS-512GT-C                        |     136.69
 ASRock C2750D4I                               |     401.98
 ASRock EP2C602-4L/D16                         |     301.99
 ASRock EP2C612 WS                             |     358.49
 ASRock Fatal1ty X299 Professional Gaming i9   |     382.98
 ASRock X299 Taichi                            |     282.98
 ASRock Z270 SuperCarrier                      |     353.98
 Asus MAXIMUS IX CODE                          |     298.98
 Asus MAXIMUS IX FORMULA                       |     388.99
 Asus MAXIMUS VIII EXTREME/ASSEMBLY            |     353.98
 Asus PRIME X299-A                             |     309.85
 Asus ROG STRIX X99 GAMING                     |     319.99
 Asus SABERTOOTH X99                           |     312.67
 Asus STRIX X299-E GAMING                      |     349.99
 Asus Sabertooth 990FX                         |     295.72
 Asus TUF X299 MARK 1                          |     339.99
 Asus VANGUARD B85                             |     287.00
 Asus X99-DELUXE II                            |     383.98
 Asus Z170-WS                                  |     338.99
 Crucial CT1050MX300SSD1                       |     267.99
 Crucial CT275MX300SSD1                        |      97.88
 Crucial CT525MX300SSD1                        |     150.99
 Crucial CT525MX300SSD4                        |     150.99
 EVGA Classified                               |     283.98
 EVGA Z270 Classified K                        |     283.98
 Gigabyte GA-X99-UD5 WIFI                      |     305.00
 Gigabyte X299 AORUS Gaming 3                  |     280.98
 Gigabyte X299 AORUS Gaming 7                  |     399.99
 Gigabyte X299 AORUS Ultra Gaming              |     343.99
 Hitachi A7K1000-1000                          |      41.99
 Hitachi HUA723020ALA640                       |      59.99
 Hitachi HUS724030ALE641                       |      65.92
 Intel DG43RK                                  |     289.79
 Kingston SA400S37/120G                        |      54.99
--More--
```

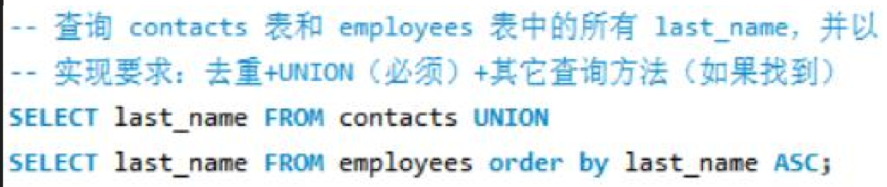


```
                 product_name                  | list_price
-----------------------------------------------+------------
 ADATA ASU800SS-128GT-C                        |      52.65
 ADATA ASU800SS-512GT-C                        |     136.69
 ASRock C2750D4I                               |     401.98
 ASRock EP2C602-4L/D16                         |     301.99
 ASRock EP2C612 WS                             |     358.49
 ASRock Fatal1ty X299 Professional Gaming i9   |     382.98
 ASRock X299 Taichi                            |     282.98
 ASRock Z270 SuperCarrier                      |     353.98
 Asus MAXIMUS IX CODE                          |     298.98
 Asus MAXIMUS IX FORMULA                       |     388.99
 Asus MAXIMUS VIII EXTREME/ASSEMBLY            |     353.98
 Asus PRIME X299-A                             |     309.85
 Asus ROG STRIX X99 GAMING                     |     319.99
 Asus SABERTOOTH X99                           |     312.67
 Asus STRIX X299-E GAMING                      |     349.99
 Asus Sabertooth 990FX                         |     295.72
 Asus TUF X299 MARK 1                          |     339.99
 Asus VANGUARD B85                             |     287.00
 Asus X99-DELUXE II                            |     383.98
 Asus Z170-WS                                  |     338.99
 Crucial CT1050MX300SSD1                       |     267.99
 Crucial CT275MX300SSD1                        |      97.88
 Crucial CT525MX300SSD1                        |     150.99
 Crucial CT525MX300SSD4                        |     150.99
 EVGA Classified                               |     283.98
 EVGA Z270 Classified K                        |     283.98
 Gigabyte GA-X99-UD5 WIFI                      |     305.00
 Gigabyte X299 AORUS Gaming 3                  |     280.98
 Gigabyte X299 AORUS Gaming 7                  |     399.99
 Gigabyte X299 AORUS Ultra Gaming              |     343.99
 Hitachi A7K1000-1000                          |      41.99
 Hitachi HUA723020ALA640                       |      59.99
 Hitachi HUS724030ALE641                       |      65.92
 Intel DG43RK                                  |     289.79
 Kingston SA400S37/120G                        |      54.99
--More--
```

（18）查询 contacts 表和 employees 表中的所有 last_name，并以 last_name 升序显示。实现要求：**去重**+UNION（必须）+其它查询方法（如果找到）

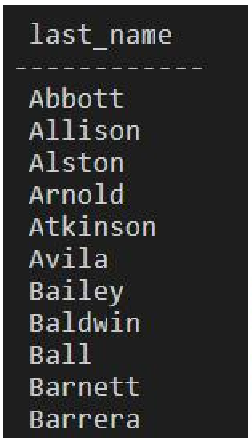


```
 last_name
-----------
 Bailey
 Boyd
 Cole
 Coleman
 Cooper
 Cox
 Crawford
 Cruz
 Dixon
 Ellis
 Fisher
```

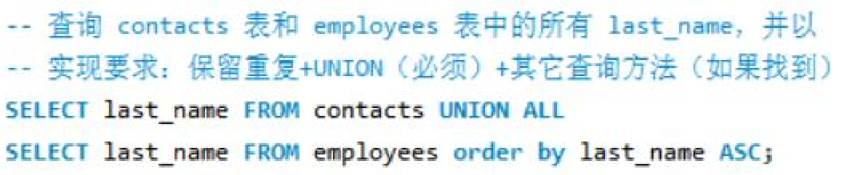


```sql
-- 查询 contacts 表和 employees 表中的所有 last_name，并以
-- 实现要求：去重+UNION（必须）+其它查询方法（如果找到）
SELECT last_name FROM contacts UNION
SELECT last_name FROM employees order by last_name ASC;
```

（19）查询 contacts 表和 employees 表中的所有 last_name，并以 last_name 升序显示。实现要求：**保留重复**+UNION ALL（必须）+其它查询方法（如果找到）

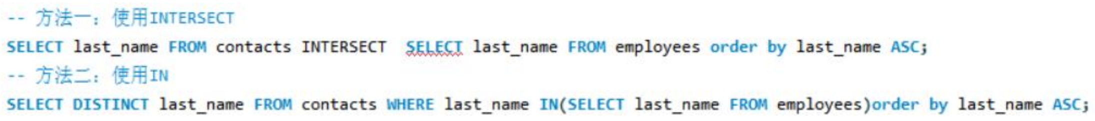


```
 last_name
------------
 Abbott
 Allison
 Alston
 Arnold
 Atkinson
 Avila
 Bailey
 Baldwin
 Ball
 Barnett
 Barrera
```

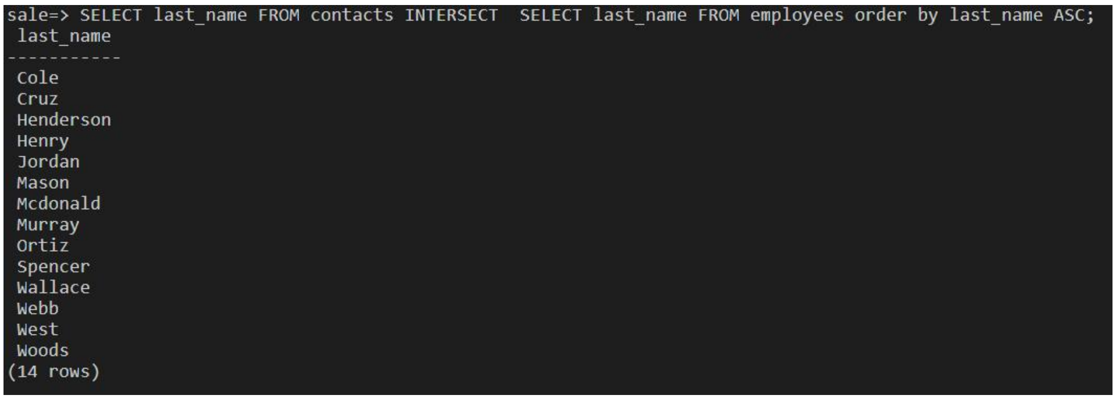


```
-- 查询 contacts 表和 employees 表中的所有 last_name, 并以
-- 实现要求：保留重复+UNION（必须）+其它查询方法（如果找到）
SELECT last_name FROM contacts UNION ALL
SELECT last_name FROM employees order by last_name ASC;
```

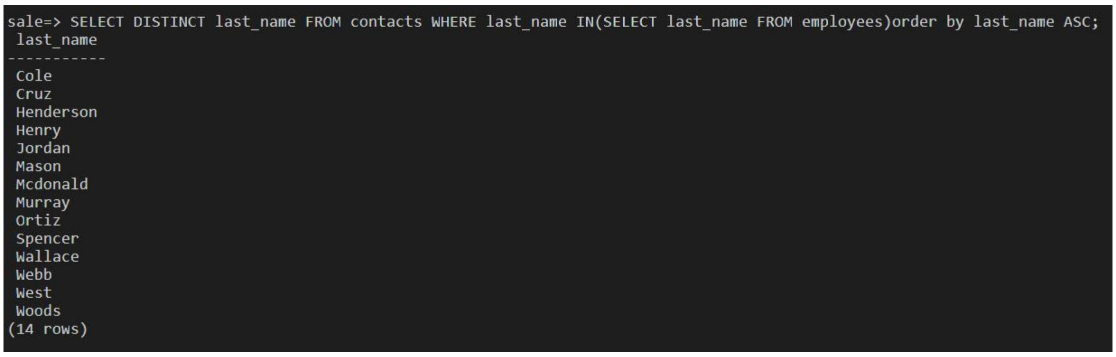
（20）查询同时出现在 contacts 表和 employees 表中的所有 last_name。实现要求：INTERSECT（必须）+其它查询方法（如果找到）



```
-- 方法一：使用INTERSECT
SELECT last_name FROM contacts INTERSECT  SELECT last_name FROM employees order by last_name ASC;
-- 方法二：使用IN
SELECT DISTINCT last_name FROM contacts WHERE last_name IN(SELECT last_name FROM employees)order by last_name ASC;
```

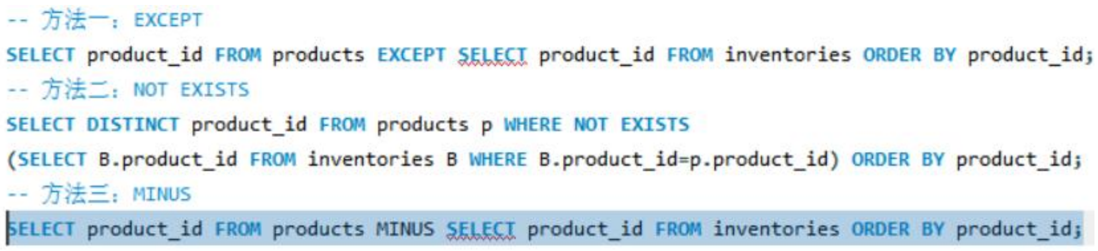


```
sale=> SELECT last_name FROM contacts INTERSECT  SELECT last_name FROM employees order by last_name ASC;
 last_name
-----------
 Cole
 Cruz
 Henderson
 Henry
 Jordan
 Mason
 Mcdonald
 Murray
 Ortiz
 Spencer
 Wallace
 Webb
 West
 Woods
(14 rows)
```

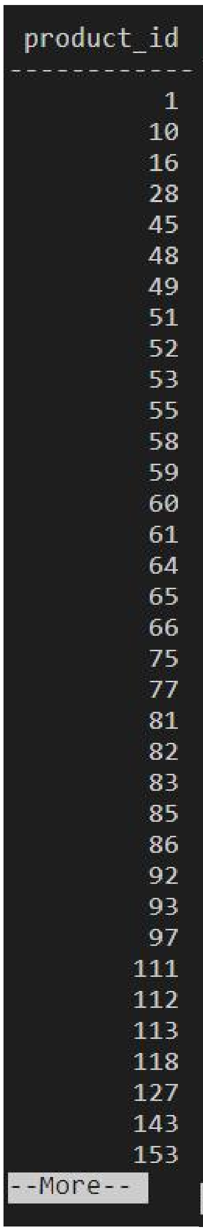


```
sale=> SELECT DISTINCT last_name FROM contacts WHERE last_name IN(SELECT last_name FROM employees)order by last_name ASC;
 last_name
-----------
 Cole
 Cruz
 Henderson
 Henry
 Jordan
 Mason
 Mcdonald
 Murray
 Ortiz
 Spencer
 Wallace
 Webb
 West
 Woods
(14 rows)
```

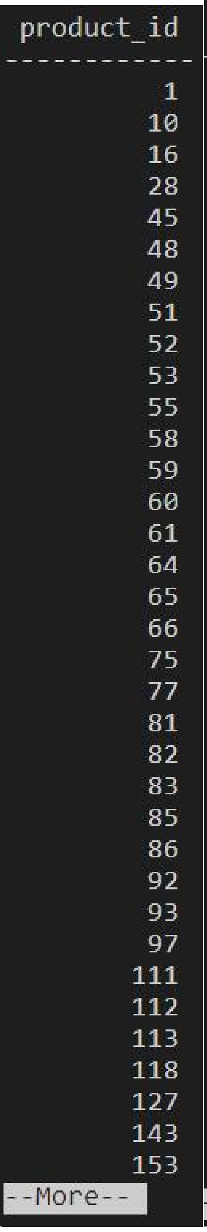
（21）查询在产品表 products 中而不在库存表 inventories 中的产品号 product_id。实现要求：MINUS/EXCEPT（必须）+其它查询方法（如果找到）



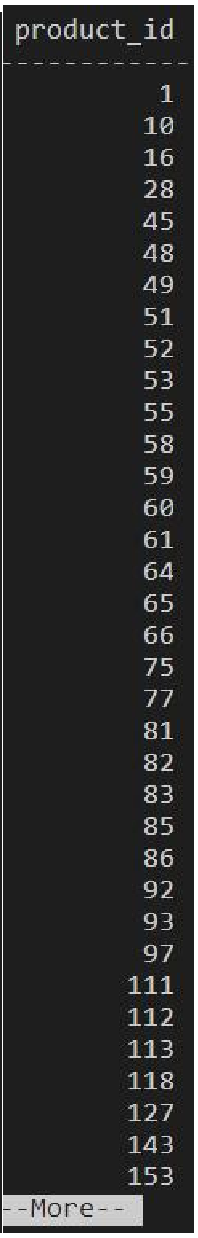
```sql
-- 方法一：EXCEPT
SELECT product_id FROM products EXCEPT SELECT product_id FROM inventories ORDER BY product_id;
-- 方法二：NOT EXISTS
SELECT DISTINCT product_id FROM products p WHERE NOT EXISTS
(SELECT B.product_id FROM inventories B WHERE B.product_id=p.product_id) ORDER BY product_id;
-- 方法三：MINUS
SELECT product_id FROM products MINUS SELECT product_id FROM inventories ORDER BY product_id;
```



```
product_id
------------
          1
         10
         16
         28
         45
         48
         49
         51
         52
         53
         55
         58
         59
         60
         61
         64
         65
         66
         75
         77
         81
         82
         83
         85
         86
         92
         93
         97
        111
        112
        113
        118
        127
        143
        153
--More--
```



```
product_id
------------
          1
         10
         16
         28
         45
         48
         49
         51
         52
         53
         55
         58
         59
         60
         61
         64
         65
         66
         75
         77
         81
         82
         83
         85
         86
         92
         93
         97
        111
        112
        113
        118
        127
        143
        153
--More--
```



```
product_id
------------
          1
         10
         16
         28
         45
         48
         49
         51
         52
         53
         55
         58
         59
         60
         61
         64
         65
         66
         75
         77
         81
         82
         83
         85
         86
         92
         93
         97
        111
        112
        113
        118
        127
        143
        153
--More--
```

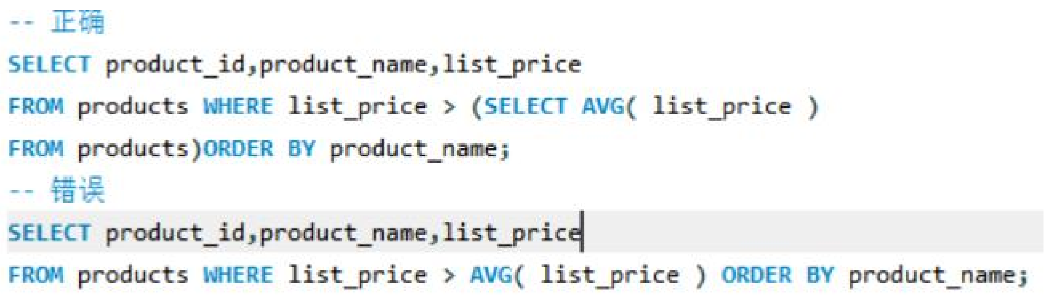
3. 实验总结

3.1 实验思考

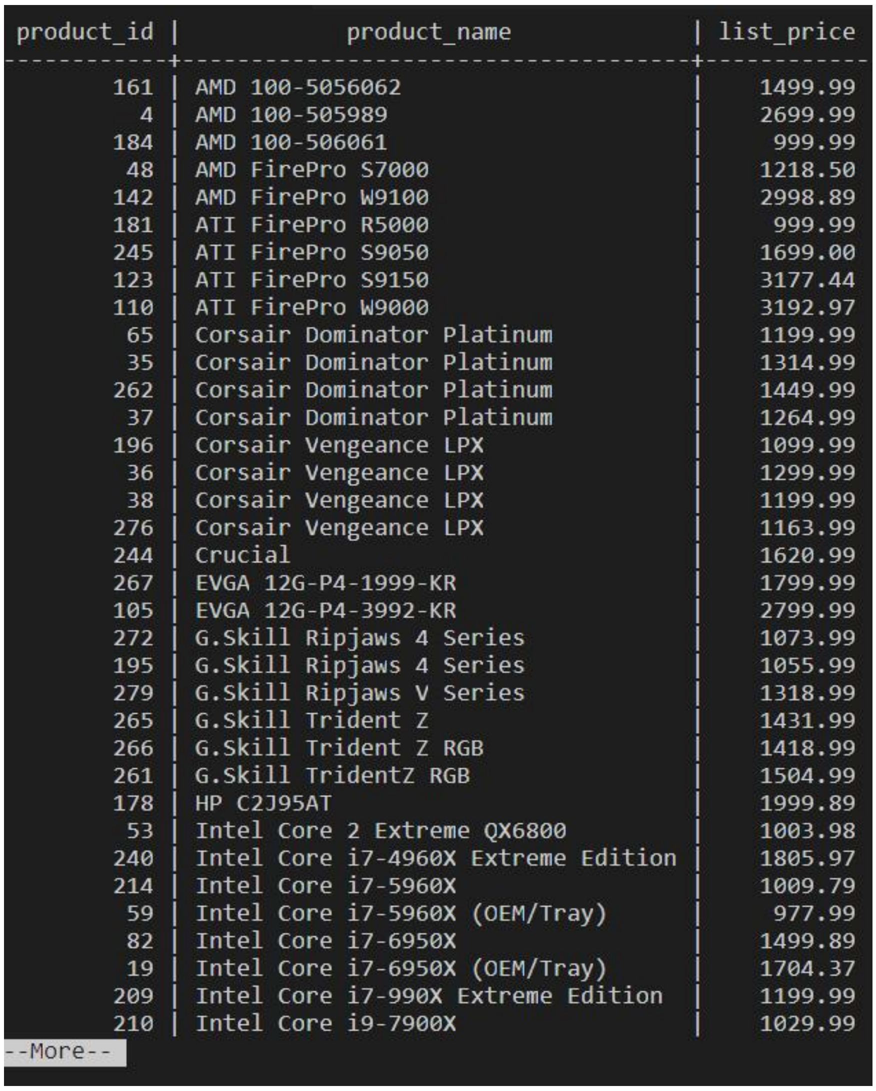
· 什么类型的查询只能用子查询实现？试举例说明。

答：Oracle 子查询可以帮助构建更具可读性的查询，并允许在不使用复杂联接或联合的情况下编写查询，允许以可以隔离每个部分的方式构建复杂的查询。什么类型的查询只能用子查询实现：

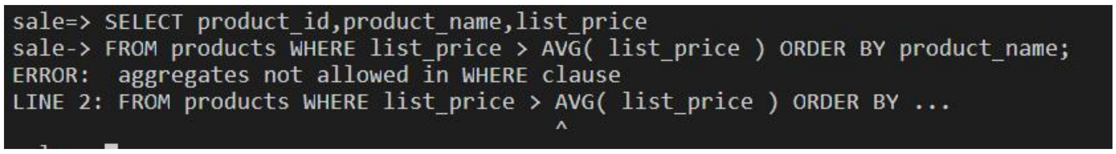
① 使用比较运算符的子查询，例如 >、>=、<、<=、<>、= 通常包括聚合函数，因为聚合函数返回单个值，可用于 WHERE 外部子句中的比较询问。比如下面的代码，如果直接在 WHERE 中使用 AVG 会出现 aggregates not allowed in WHERE clause 的错误，而通过子查询就可以完美地解决这个问题。



```
-- 正确
SELECT product_id,product_name,list_price
FROM products WHERE list_price > (SELECT AVG( list_price )
FROM products)ORDER BY product_name;
-- 错误
SELECT product_id,product_name,list_price
FROM products WHERE list_price > AVG( list_price ) ORDER BY product_name;
```

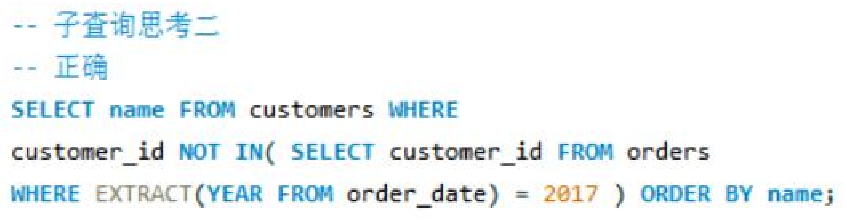


```
 product_id |                product_name                | list_price
------------+--------------------------------------------+------------
        161 | AMD 100-5056062                            |    1499.99
          4 | AMD 100-505989                             |    2699.99
        184 | AMD 100-506061                             |     999.99
         48 | AMD FirePro S7000                          |    1218.50
        142 | AMD FirePro W9100                          |    2998.89
        181 | ATI FirePro R5000                          |     999.99
        245 | ATI FirePro S9050                          |    1699.00
        123 | ATI FirePro S9150                          |    3177.44
        110 | ATI FirePro W9000                          |    3192.97
         65 | Corsair Dominator Platinum                 |    1199.99
         35 | Corsair Dominator Platinum                 |    1314.99
        262 | Corsair Dominator Platinum                 |    1449.99
         37 | Corsair Dominator Platinum                 |    1264.99
        196 | Corsair Vengeance LPX                      |    1099.99
         36 | Corsair Vengeance LPX                      |    1299.99
         38 | Corsair Vengeance LPX                      |    1199.99
        276 | Corsair Vengeance LPX                      |    1163.99
        244 | Crucial                                    |    1620.99
        267 | EVGA 12G-P4-1999-KR                        |    1799.99
        105 | EVGA 12G-P4-3992-KR                        |    2799.99
        272 | G.Skill Ripjaws 4 Series                   |    1073.99
        195 | G.Skill Ripjaws 4 Series                   |    1055.99
        279 | G.Skill Ripjaws V Series                   |    1318.99
        265 | G.Skill Trident Z                          |    1431.99
        266 | G.Skill Trident Z RGB                      |    1418.99
        261 | G.Skill TridentZ RGB                       |    1504.99
        178 | HP C2J95AT                                 |    1999.89
         53 | Intel Core 2 Extreme QX6800                |    1003.98
        240 | Intel Core i7-4960X Extreme Edition        |    1805.97
        214 | Intel Core i7-5960X                        |    1009.79
         59 | Intel Core i7-5960X (OEM/Tray)             |     977.99
         82 | Intel Core i7-6950X                        |    1499.89
         19 | Intel Core i7-6950X (OEM/Tray)             |    1704.37
        209 | Intel Core i7-990X Extreme Edition         |    1199.99
        210 | Intel Core i9-7900X                        |    1029.99
--More--
```

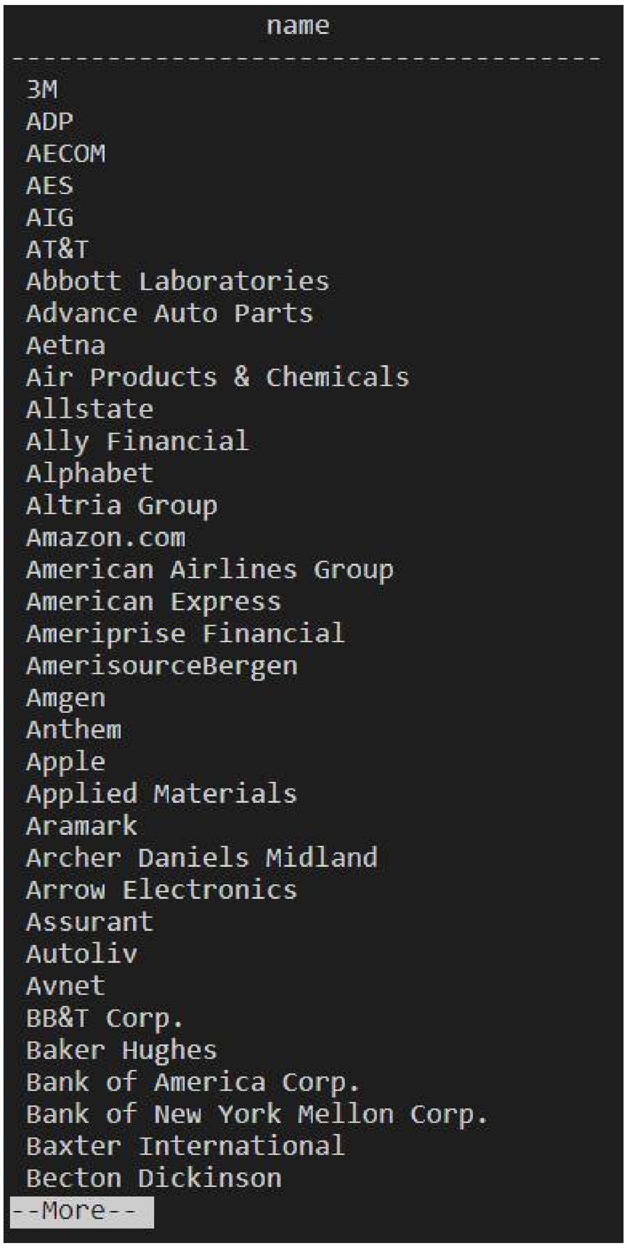


```
sale=> SELECT product_id,product_name,list_price
sale-> FROM products WHERE list_price > AVG( list_price ) ORDER BY product_name;
ERROR:  aggregates not allowed in WHERE clause
LINE 2: FROM products WHERE list_price > AVG( list_price ) ORDER BY ...
                                         ^
```

② 还有 NOT　IN 等情况，比如下面的代码，用于查找 2017 年尚未下订单的所有客户。使用 IN 运算符的子查询通常返回零个或多个值的列表。子查询返回结果集后，外部查询使用它们。



```
-- 子查询思考二
-- 正确
SELECT name FROM customers WHERE
customer_id NOT IN( SELECT customer_id FROM orders
WHERE EXTRACT(YEAR FROM order_date) = 2017 ) ORDER BY name;
```

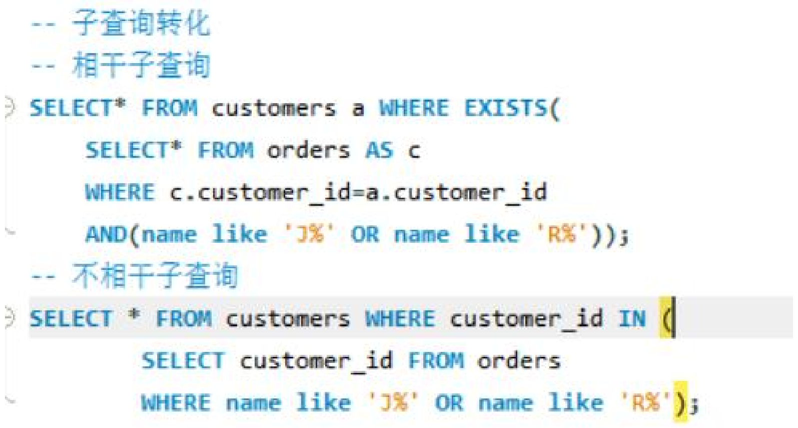


```
                name
------------------------------------
 3M
 ADP
 AECOM
 AES
 AIG
 AT&T
 Abbott Laboratories
 Advance Auto Parts
 Aetna
 Air Products & Chemicals
 Allstate
 Ally Financial
 Alphabet
 Altria Group
 Amazon.com
 American Airlines Group
 American Express
 Ameriprise Financial
 AmerisourceBergen
 Amgen
 Anthem
 Apple
 Applied Materials
 Aramark
 Archer Daniels Midland
 Arrow Electronics
 Assurant
 Autoliv
 Avnet
 BB&T Corp.
 Baker Hughes
 Bank of America Corp.
 Bank of New York Mellon Corp.
 Baxter International
 Becton Dickinson
--More--
```

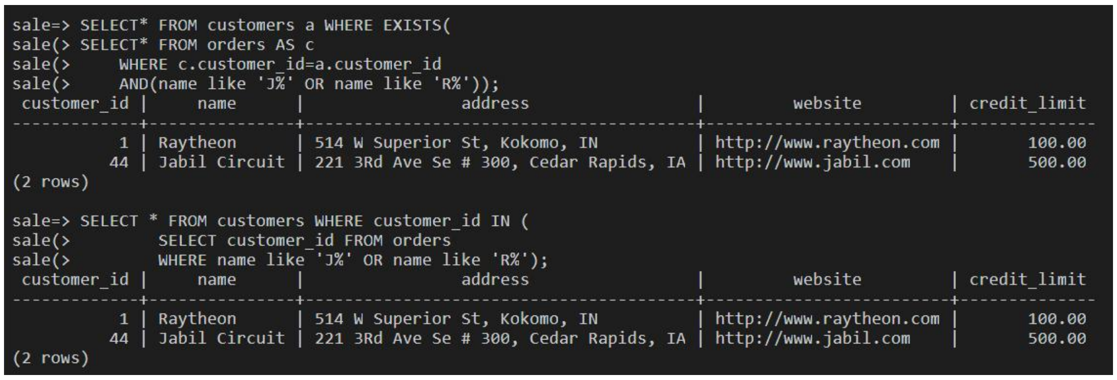
· 相关子查询与不相关子查询的区别？什么情形下使用相关子查询？如何将相关子查询转换成一般查询？（说明：一般查询指不一定必须使用子查询）

答：相关子查询是一个子查询，其某些子句引用外部查询中的列表达式。不相关子查询返回结果集后，外部查询使用它们。换句话说，外部查询依赖于子查询。但是，子查询是独立的，不依赖于外部查询的值。与不相关子查询不同，相关子查询是使用来自外部查询的值的子

查询。可以为外部查询选择的每一行评估一次相关子查询。因此，使用相关子查询的查询可能会很慢。以下面代码为例，可将相干子查询转换成不相干子查询。



```
-- 子查询转化
-- 相干子查询
SELECT* FROM customers a WHERE EXISTS(
    SELECT* FROM orders AS c
    WHERE c.customer_id=a.customer_id
    AND(name like 'J%' OR name like 'R%'));
-- 不相干子查询
SELECT * FROM customers WHERE customer_id IN (
        SELECT customer_id FROM orders
        WHERE name like 'J%' OR name like 'R%');
```



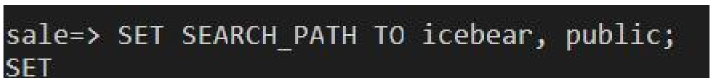
```
sale=> SELECT* FROM customers a WHERE EXISTS(
sale(> SELECT* FROM orders AS c
sale(>     WHERE c.customer_id=a.customer_id
sale(>     AND(name like 'J%' OR name like 'R%'));
 customer_id |     name      |                address                 |         website          | credit_limit
-------------+---------------+----------------------------------------+--------------------------+-------------
           1 | Raytheon      | 514 W Superior St, Kokomo, IN          | http://www.raytheon.com |       100.00
          44 | Jabil Circuit | 221 3Rd Ave Se # 300, Cedar Rapids, IA | http://www.jabil.com    |       500.00
(2 rows)

sale=> SELECT * FROM customers WHERE customer_id IN (
sale(>         SELECT customer_id FROM orders
sale(>         WHERE name like 'J%' OR name like 'R%');
 customer_id |     name      |                address                 |         website          | credit_limit
-------------+---------------+----------------------------------------+--------------------------+-------------
           1 | Raytheon      | 514 W Superior St, Kokomo, IN          | http://www.raytheon.com |       100.00
          44 | Jabil Circuit | 221 3Rd Ave Se # 300, Cedar Rapids, IA | http://www.jabil.com    |       500.00
(2 rows)
```

3.2 对实验的认识

通过实验我对 openGauss 中的一些语句更熟悉了。如 SET SEARCH_PATH TO icebear, public;可以将搜索路径设置为 icebear、public，首先搜索 icebear。如 SELECT * FROM customer_t1;可以用来查询表 customer_t1 的所有数据。gsql -d sale -p 26000 -U yuxiaoping -W yuxiaoping@123   -r 可以用来将新用户连接到数据库。可以使用 gsql -d postgres -p 26000 -r 连接到 postgres。gs_om -t start 可以开启数据库。

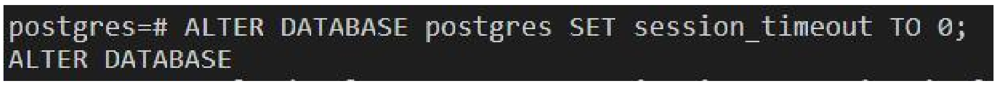
3.3 遇到的困难及解决方法

要更改当前会话的默认 Schema，请使用 SET 命令。执行如下命令 SET SEARCH_PATH To icebear,public;将搜索路径设置为 myschema、public，首先搜索 myschema。



```
sale=> SET SEARCH_PATH TO icebear, public;
SET
```

高斯默认有 session 超时时间，若想要 session 一直保持，需要修改配置项：ALTER DATABASE sale SET session_timeout TO 0;



```
postgres=# ALTER DATABASE postgres SET session_timeout TO 0;
ALTER DATABASE
```