数据查询与修改

1.3.1 单表查询

查询1:

从订单表 `ORDERS` 中，找出由收银员 `Clerk#000000951` 处理的满足下列条件的所有订单 `O_ORDERKEY`：

（1）订单总价位于 `[起始价格 5000, 结束价格 100000]`

（2）下单日期在 `开始日期 2019-01-02 00:00:00` 至 `结束日期 2020-08-31 00:00:00` 之间，

（3）订单状态 `O_ORDERSTATUS` 不为空

列出这些订单的订单 `key`（`O_ORDERKEY`）、客户 `key`、订单状态、订单总价、下单日期（重命名为 `O_DATE`）、订单优先级和发货优先级；

要求：对查询结果，按照订单优先级从高到低、发货优先级从高到低排序。

```sql
-- 查询订单信息
SELECT
    O_ORDERKEY,
    O_CUSTKEY,
    O_ORDERSTATUS,
    O_TOTALPRICE,
    O_ORDERDATE AS O_DATE,
    O_ORDERPRIORITY,
    O_SHIPPRIORITY
FROM
    ORDERS
WHERE
    O_CLERK = 'Clerk#000000951'
    AND O_TOTALPRICE BETWEEN 5000 AND 100000
    AND O_ORDERDATE BETWEEN '2019-01-02' AND '2020-08-31'
    AND O_ORDERSTATUS IS NOT NULL
ORDER BY
    O_ORDERPRIORITY DESC,
    O_SHIPPRIORITY DESC;
```

```
omm=# SELECT O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE, O_ORDERDATE AS O_DATE, O_ORDERPRIORITY, O_SHIPPRIORITY
FROM ORDERS
WHERE O_CLERK = 'Clerk#000000951';omm-# omm-#
 o_orderkey | o_custkey | o_orderstatus | o_totalprice |       o_date        | o_orderpriority | o_shippriority
------------+-----------+---------------+--------------+---------------------+-----------------+----------------
          1 |      7381 | O             |    181585.13 | 2019-01-02 00:00:00 | 5-LOW           |              0
        839 |      5578 | O             |    104005.14 | 2018-08-08 00:00:00 | 1-URGENT        |              0
       2338 |     27874 | O             |     22264.72 | 2020-09-15 00:00:00 | 2-HIGH          |              0
       4579 |     20828 | O             |    147919.32 | 2018-12-01 00:00:00 | 2-HIGH          |              0
       8452 |     27832 | F             |    147102.45 | 2015-07-31 00:00:00 | 4-NOT SPECIFIED |              0
       9185 |      2893 | F             |     92840.21 | 2017-06-16 00:00:00 | 2-HIGH          |              0
      12163 |      1733 | O             |    183726.95 | 2020-07-20 00:00:00 | 5-LOW           |              0
      13508 |     16033 | O             |     42756.76 | 2020-04-17 00:00:00 | 4-NOT SPECIFIED |              0
      14277 |     18920 | O             |    133599.91 | 2021-02-14 00:00:00 | 4-NOT SPECIFIED |              0
      15073 |      1468 | F             |    138584.35 | 2015-01-26 00:00:00 | 3-MEDIUM        |              0
      17636 |     15205 | F             |    137295.03 | 2017-02-05 00:00:00 | 5-LOW           |              0
      19200 |       854 | O             |    144151.90 | 2020-07-27 00:00:00 | 4-NOT SPECIFIED |              0
      19205 |     11662 | F             |    327627.84 | 2016-07-27 00:00:00 | 3-MEDIUM        |              0
      20547 |     16147 | F             |     38376.91 | 2016-08-27 00:00:00 | 2-HIGH          |              0
      21312 |      5638 | O             |     55741.75 | 2019-02-01 00:00:00 | 1-URGENT        |              0
      25639 |     25135 | F             |     46746.31 | 2017-10-12 00:00:00 | 3-MEDIUM        |              0
      26885 |     25732 | O             |    226010.66 | 2020-05-11 00:00:00 | 3-MEDIUM        |              0
      27364 |      4489 | O             |    201233.85 | 2018-05-21 00:00:00 | 3-MEDIUM        |              0
      40932 |     23078 | O             |    176808.89 | 2021-05-12 00:00:00 | 3-MEDIUM        |              0

```

```
      42817 |     21661 | O             |     44438.99 | 2020-08-31 00:00:
00 | 5-LOW           |             0
      47142 |     19433 | O             |     73866.30 | 2019-03-24 00:00:
00 | 1-URGENT        |             0
      60419 |     21305 | F             |    114536.40 | 2015-06-20 00:00:
00 | 4-NOT SPECIFIED |             0
      60867 |     22738 | O             |     92114.35 | 2020-07-27 00:00:
00 | 3-MEDIUM        |             0
      64612 |     20027 | F             |    307272.91 | 2016-12-26 00:00:
00 | 4-NOT SPECIFIED |             0
      66470 |      1373 | F             |     68381.45 | 2016-05-10 00:00:
00 | 3-MEDIUM        |             0
      66531 |     23941 | F             |    101155.27 | 2015-10-20 00:00:
00 | 3-MEDIUM        |             0
      84197 |     28901 | O             |     79241.84 | 2019-05-22 00:00:
00 | 5-LOW           |             0
      95623 |     20209 | O             |     99889.91 | 2020-12-10 00:00:
00 | 2-HIGH          |             0
      96870 |     13928 | F             |    117595.25 | 2015-01-07 00:00:
00 | 1-URGENT        |             0
     100611 |      6268 | O             |    119354.28 | 2019-02-14 00:00:
00 | 5-LOW           |             0
     105920 |      4939 | F             |    113306.11 | 2016-06-20 00:00:
00 | 2-HIGH          |             0
     112965 |       472 | F             |     15036.22 | 2016-08-11 00:00:
00 | 3-MEDIUM        |             0
     114599 |     28057 | F             |     61960.06 | 2015-04-24 00:00:
00 | 4-NOT SPECIFIED |             0
```

查询2：

从订单明细表 `LINEITEM` 中，找出满足下列条件的所有订单 `L_ORDERKEY`：

（1）数量位于 `[起始数量 30, 结束数量 50]`，

（2）退货标志为 `'N'` 的订单中，价格不小于 `最低价格 20000`

列出这些订单的 `L_ORDERKEY`、`L_SUPPKEY`、`L_EXTENDEDPRICE`；要求：对查询结果，按照价格从高到低排序，并且对查询结果使用 `DISTINCT` 去重。

比较对查询结果去重和不去重，在查询时间和查询结果上的差异。

```sql
-- 去重查询
EXPLAIN ANALYZE
SELECT DISTINCT L_ORDERKEY, L_SUPPKEY, L_EXTENDEDPRICE
FROM LINEITEM
WHERE L_QUANTITY BETWEEN 30 AND 50
  AND L_RETURNFLAG = 'N'
  AND L_EXTENDEDPRICE >= 96000
ORDER BY L_EXTENDEDPRICE DESC;

-- 不去重查询
EXPLAIN ANALYZE
SELECT L_ORDERKEY, L_SUPPKEY, L_EXTENDEDPRICE
FROM LINEITEM
WHERE L_QUANTITY BETWEEN 30 AND 50
  AND L_RETURNFLAG = 'N'
  AND L_EXTENDEDPRICE >= 96000
ORDER BY L_EXTENDEDPRICE DESC;
```

去重

查询结果

```
omm=# SELECT DISTINCT L_ORDERKEY, L_SUPPKEY, L_EXTENDEDPRICE
FROM LINEITEM
WHERE L_QUANTITY BETWEEN 30 AND 50
  AND L_RETURNFLAG = 'N'
  AND L_EXTENDEDPRICE >= 96000
ORDER BY L_EXTENDEDPRICE DESC;omm-# omm-# omm-# omm-# omm-#
 l_orderkey | l_suppkey | l_extendedprice
------------+-----------+-----------------
     549057 |      2000 |        96949.50
     272229 |       557 |        96899.50
     705441 |      1999 |        96899.50
     719041 |      1519 |        96899.50
    1028352 |      1036 |        96849.50
     814209 |      1999 |        96799.50
    1114084 |       517 |        96799.50
     111329 |      1553 |        96749.50
     456194 |        35 |        96749.50
     747776 |      1552 |        96749.50
     163712 |       516 |        96699.50
     323143 |      1516 |        96699.50
      10308 |       999 |        96649.50
     330503 |      1032 |        96649.50
     916000 |      1998 |        96649.50
     ........
          244676 |       506 |        96149.00
     428514 |      1984 |        96149.00
     868550 |      1538 |        96149.00
    1101380 |        23 |        96149.00
     293607 |       537 |        96099.50
     400421 |      1509 |        96099.50
     787011 |       537 |        96099.50
    1096550 |       533 |        96099.50
     265669 |      1538 |        96099.00
     368230 |       540 |        96099.00
     787937 |       505 |        96099.00
     892805 |       984 |        96099.00
     933921 |      1021 |        96099.00
     603969 |       991 |        96049.50
     177411 |       505 |        96049.00
     334182 |      1535 |        96049.00
     591331 |      1020 |        96049.00
     591716 |      1537 |        96049.00
     610368 |       502 |        96049.00
(70 rows)

```

查询时间：223.757 ms

```
omm=# EXPLAIN ANALYZE
SELECT DISTINCT L_ORDERKEY, L_SUPPKEY, L_EXTENDEDPRICE
FROM LINEITEM
WHERE L_QUANTITY BETWEEN 30 AND 50
  AND L_RETURNFLAG = 'N'
  AND L_EXTENDEDPRICE >= 96000
ORDER BY L_EXTENDEDPRICE DESC;omm-# omm-# omm-# omm-# omm-# omm-#

                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 Unique  (cost=48915.99..48916.25 rows=26 width=16) (actual time=223.653..223.662 rows=70 loops=1)
   ->  Sort  (cost=48915.99..48916.06 rows=26 width=16) (actual time=223.652..223.655 rows=70 loops=1)
         Sort Key: l_extendedprice DESC, l_orderkey, l_suppkey
         Sort Method: quicksort  Memory: 29kB
         ->  Seq Scan on lineitem  (cost=0.00..48915.38 rows=26 width=16) (actual time=5.454..223.591 rows=70 loops=1)
               Filter: ((l_quantity >= 30::numeric) AND (l_quantity <= 50::numeric) AND (l_extendedprice >= 96000::numeric) AND (l_returnflag = 'N'::bpchar))
               Rows Removed by Filter: 1199899
 Total runtime: 223.757 ms
```

不去重

查询结果

```
omm=# SELECT L_ORDERKEY, L_SUPPKEY, L_EXTENDEDPRICE
FROM LINEITEM
WHERE L_QUANTITY BETWEEN 30 AND 50
  AND L_RETURNFLAG = 'N'
  AND L_EXTENDEDPRICE >= 96000
ORDER BY L_EXTENDEDPRICE DESC;omm-# omm-# omm-# omm-# omm-#
 l_orderkey | l_suppkey | l_extendedprice
------------+-----------+-----------------
     549057 |      2000 |        96949.50
     272229 |       557 |        96899.50
     719041 |      1519 |        96899.50
     705441 |      1999 |        96899.50
    1028352 |      1036 |        96849.50
     814209 |      1999 |        96799.50
    1114084 |       517 |        96799.50
     456194 |        35 |        96749.50
     747776 |      1552 |        96749.50
     111329 |      1553 |        96749.50
     323143 |      1516 |        96699.50
  ........
     578660 |      1509 |        96199.50
       8134 |       988 |        96199.00
     809378 |      1022 |        96149.50
     965285 |      1535 |        96149.50
    1075681 |      1510 |        96149.50
    1101380 |        23 |        96149.00
     868550 |      1538 |        96149.00
     244676 |       506 |        96149.00
     428514 |      1984 |        96149.00
     400421 |      1509 |        96099.50
     787011 |       537 |        96099.50
     293607 |       537 |        96099.50
    1096550 |       533 |        96099.50
     933921 |      1021 |        96099.00
     892805 |       984 |        96099.00
     787937 |       505 |        96099.00
     265669 |      1538 |        96099.00
     368230 |       540 |        96099.00
     603969 |       991 |        96049.50
     610368 |       502 |        96049.00
     591716 |      1537 |        96049.00
     591331 |      1020 |        96049.00
     334182 |      1535 |        96049.00
     177411 |       505 |        96049.00
(70 rows)
```

查询时间：223.187 ms

```
omm=# EXPLAIN ANALYZE
SELECT L_ORDERKEY, L_SUPPKEY, L_EXTENDEDPRICE
FROM LINEITEM
WHERE L_QUANTITY BETWEEN 30 AND 50
  AND L_RETURNFLAG = 'N'
  AND L_EXTENDEDPRICE >= 96000
ORDER BY L_EXTENDEDPRICE DESC;omm-# omm-# omm-# omm-# omm-# omm-#
                                                                QUE
RY PLAN
------------------------------------------------------------------------------
------------------------------------------------------------------------------
----
 Sort  (cost=48915.99..48916.06 rows=26 width=16) (actual time=223.107..22
3.110 rows=70 loops=1)
   Sort Key: l_extendedprice DESC
   Sort Method: quicksort  Memory: 29kB
   ->  Seq Scan on lineitem  (cost=0.00..48915.38 rows=26 width=16) (actua
l time=5.467..223.043 rows=70 loops=1)
         Filter: ((l_quantity >= 30::numeric) AND (l_quantity <= 50::numer
ic) AND (l_extendedprice >= 96000::numeric) AND (l_returnflag = 'N'::bpcha
r))
         Rows Removed by Filter: 1199899
 Total runtime: 223.187 ms
(7 rows)
```

1.3.2 字符串操作

查询3：

从客户表 `CUSTOMER` 中，找出满足下列条件的客户：

（1）客户电话开头部分包含 `'10'`，或者客户市场领域中包含 `'BUILDING'`，并且

（2）客户电话结尾不为 `'8'`

```
SELECT C_CUSTKEY, C_NAME
FROM CUSTOMER
WHERE (C_PHONE LIKE '10%' OR C_MKTSEGMENT LIKE '%BUILDING%')
  AND C_PHONE NOT LIKE '%8';
```

实验结果

```
 c_custkey |       c_name
-----------+--------------------
         2 | Customer#000000002
         3 | Customer#000000003
         4 | Customer#000000004
         5 | Customer#000000005
         6 | Customer#000000006
         7 | Customer#000000007
         8 | Customer#000000008
         9 | Customer#000000009
        10 | Customer#000000010
.......
      1955 | Customer#000001955
      1956 | Customer#000001956
      1958 | Customer#000001958
      1959 | Customer#000001959
      1960 | Customer#000001960
      1962 | Customer#000001962
      1963 | Customer#000001963
      1964 | Customer#000001964
      1965 | Customer#000001965
      1966 | Customer#000001966
      1967 | Customer#000001967
.......
```

查询4：

从客户表 `CUSTOMER` 中，找出满足下列条件的客户姓名：

（1）客户 `key` 由 2 个字符组成

（2）客户地址至少包括 18 个字符，即地址字符串的长度不小于 18。

```sql
SELECT C_NAME
FROM CUSTOMER
WHERE C_CUSTKEY::TEXT LIKE '__'  -- 假设C_CUSTKEY为整数，需要转换为文本比较
  AND LENGTH(C_ADDRESS) >= 18;
```

实验结果

```
        c_name
--------------------
 Customer#000000010
 Customer#000000012
 Customer#000000013
 Customer#000000014
 Customer#000000015
 Customer#000000017
 Customer#000000018
 Customer#000000019
 Customer#000000020
 Customer#000000022
 Customer#000000023
 Customer#000000024
 .......
 Customer#000000084
 Customer#000000085
 Customer#000000087
 Customer#000000088
 Customer#000000089
 Customer#000000091
 Customer#000000092
 Customer#000000093
 Customer#000000094
 Customer#000000095
 Customer#000000096
 Customer#000000097
 Customer#000000098
 Customer#000000099
(64 rows)
```

1.3.3 集合操作

查询5：

使用集合并操作 `UNION` 、 `UNION ALL` ，从订单明细表 `LINEITEM` 查询满足下列条件的订单 `L_ORDERKEY` ：

（1）订单发货日期早于 `'2016-01-01'` ，或者

（2）订单数量大于 `100`

对比 `UNION ALL` 、 `UNION` 操作在查询结果、执行时间上的差异。

```sql
-- 使用 UNION ALL
EXPLAIN ANALYZE
SELECT L_ORDERKEY
FROM LINEITEM
WHERE L_SHIPDATE < '2016-01-01'::DATE

UNION ALL

SELECT L_ORDERKEY
FROM LINEITEM
WHERE L_QUANTITY > 100;

-- 使用 UNION
EXPLAIN ANALYZE
SELECT L_ORDERKEY
FROM LINEITEM
WHERE L_SHIPDATE < '2016-01-01'::DATE

UNION

SELECT L_ORDERKEY
FROM LINEITEM
WHERE L_QUANTITY > 100;
```

使用 UNION ALL

实验结果

```
 l_orderkey
------------
          6
         37
         37
         37
        128
        129
        129
...........
       1504
       1504
       1505
       1505
       1506
       1506
       1506
       1506
       1506
       1537
       1537
...........
       5218
       5220
       5254
       5254
       5254
       5254
       5254
       5254
...........
```

执行时间: 365.901 ms

```
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Result  (cost=0.00..81345.32 rows=151409 width=4) (actual time=0.057..361.636 rows=151587 loops=1)
   ->  Append  (cost=0.00..81345.32 rows=151409 width=4) (actual time=0.056..352.949 rows=151587 loops=1)
         ->  Seq Scan on lineitem  (cost=0.00..39915.61 rows=151408 width=4) (actual time=0.055..167.003 rows=151587 loops=1)
               Filter: (l_shipdate < '2016-01-01 00:00:00'::timestamp(0) without time zone)
               Rows Removed by Filter: 1048382
         ->  Seq Scan on lineitem  (cost=0.00..39915.61 rows=1 width=4) (actual time=178.241..178.241 rows=0 loops=1)
               Filter: (l_quantity > 100::numeric)
               Rows Removed by Filter: 1199969
 Total runtime: 365.901 ms
(9 rows)
```

使用 UNION

实验结果

```
 l_orderkey
------------
     865029
     370596
     310753
     363111
    1096454
     728802
.....
     834023
     733601
     344993
     970624
     104033
     738276
     821126
    1147808
     551749
    1083878
     361732
.....
```

执行时间: 364.971 ms

```
omm=# EXPLAIN ANALYZE
SELECT L_ORDERKEY
FROM LINEITEM
WHERE L_SHIPDATE < '2016-01-01'::DATE

UNION

SELECT L_ORDERKEY
FROM LINEITEM
WHERE L_QUANTITY > 100;omm-# omm-# omm-# omm-# omm-# omm-# omm-# omm-# omm
-#
                                                                 QUERY PLAN
-----------------------------------------------------------------------------------------------------
 HashAggregate  (cost=81723.84..83237.93 rows=151409 width=4) (actual time
=359.014..363.225 rows=41621 loops=1)
   Group By Key: public.lineitem.l_orderkey
   ->  Append  (cost=0.00..81345.32 rows=151409 width=4) (actual time=0.06
9..337.867 rows=151587 loops=1)
         ->  Seq Scan on lineitem  (cost=0.00..39915.61 rows=151408 width=
4) (actual time=0.066..155.120 rows=151587 loops=1)
               Filter: (l_shipdate < '2016-01-01 00:00:00'::timestamp(0) w
ithout time zone)
               Rows Removed by Filter: 1048382
         ->  Seq Scan on lineitem  (cost=0.00..39915.61 rows=1 width=4) (a
ctual time=174.747..174.747 rows=0 loops=1)
               Filter: (l_quantity > 100::numeric)
               Rows Removed by Filter: 1199969
 Total runtime: 364.971 ms
(10 rows)
```

查询6：

结合教材 3.4.1 节元组变量样例，使用集合操作 `EXCEPT` 、 `EXCEPT ALL` ，从供应商表 `SUPPLIER` 中，查询账户余额最大的供应商。

对比使用 `EXCEPT` 、 `EXCEPT ALL` 、聚集函数 `MAX` ，完成此查询在执行时间、查询结果上的异同。

```sql
-- 更新统计信息
ANALYZE supplier;

-- 使用 EXCEPT
EXPLAIN ANALYZE
SELECT s_suppkey, s_name
FROM supplier
EXCEPT
(
    SELECT t1.s_suppkey, t1.s_name
    FROM supplier t1
    JOIN supplier t2 ON t1.s_acctbal < t2.s_acctbal
);

-- 使用 EXCEPT ALL
EXPLAIN ANALYZE
SELECT s_suppkey, s_name
FROM supplier
EXCEPT ALL
(
    SELECT t1.s_suppkey, t1.s_name
    FROM supplier t1
    JOIN supplier t2 ON t1.s_acctbal < t2.s_acctbal
);

-- 使用 MAX 聚集函数
EXPLAIN ANALYZE
SELECT s_suppkey, s_name
FROM supplier
WHERE s_acctbal = (
    SELECT MAX(s_acctbal)
    FROM supplier
);
```

使用 EXCEPT

实验结果

```
 s_suppkey |          s_name
-----------+--------------------------
       892 | Supplier#000000892
(1 row)
```

执行时间: 1001.204 ms

```
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 HashSetOp Except  (cost=0.00..80220.99 rows=2000 width=30) (actual time=1001.026..1001.039 rows=1 loops=1)
   ->  Append  (cost=0.00..73544.33 rows=1335333 width=30) (actual time=0.018..788.066 rows=2000997 loops=1)
         ->  Subquery Scan on "*SELECT* 1"  (cost=0.00..82.00 rows=2000 width=30) (actual time=0.017..1.341 rows=2000 loops=1)
               ->  Seq Scan on supplier  (cost=0.00..62.00 rows=2000 width=30) (actual time=0.014..0.802 rows=2000 loops=1)
         ->  Subquery Scan on "*SELECT* 2"  (cost=0.00..73462.33 rows=1333333 width=30) (actual time=0.034..700.892 rows=1998997 loops=1)
               ->  Nested Loop  (cost=0.00..60129.00 rows=1333333 width=30) (actual time=0.033..586.081 rows=1998997 loops=1)
                     Join Filter: (t1.s_acctbal < t2.s_acctbal)
                     Rows Removed by Join Filter: 2001003
                     ->  Seq Scan on supplier t1  (cost=0.00..62.00 rows=2000 width=36) (actual time=0.004..0.151 rows=2000 loops=1)
                     ->  Materialize  (cost=0.00..72.00 rows=2000 width=6) (actual time=0.065..111.885 rows=4000000 loops=2000)
                           ->  Seq Scan on supplier t2  (cost=0.00..62.00 rows=2000 width=6) (actual time=0.004..0.636 rows=2000 loops=1)
 Total runtime: 1001.204 ms
(12 rows)
```

使用 EXCEPT ALL

实验结果

```
 s_suppkey |          s_name
-----------+---------------------------
       892 | Supplier#000000892
(1 row)
```

执行时间: 995.772 ms

```
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 HashSetOp Except All  (cost=0.00..80220.99 rows=2000 width=30) (actual time=995.580..995.594 rows=1 loops=1)
   ->  Append  (cost=0.00..73544.33 rows=1335333 width=30) (actual time=0.020..783.643 rows=2000997 loops=1)
         ->  Subquery Scan on "*SELECT* 1"  (cost=0.00..82.00 rows=2000 width=30) (actual time=0.020..1.236 rows=2000 loops=1)
               ->  Seq Scan on supplier  (cost=0.00..62.00 rows=2000 width=30) (actual time=0.017..0.711 rows=2000 loops=1)
         ->  Subquery Scan on "*SELECT* 2"  (cost=0.00..73462.33 rows=1333333 width=30) (actual time=0.032..697.765 rows=1998997 loops=1)
               ->  Nested Loop  (cost=0.00..60129.00 rows=1333333 width=30) (actual time=0.032..583.712 rows=1998997 loops=1)
                     Join Filter: (t1.s_acctbal < t2.s_acctbal)
                     Rows Removed by Join Filter: 2001003
                     ->  Seq Scan on supplier t1  (cost=0.00..62.00 rows=2000 width=36) (actual time=0.004..0.189 rows=2000 loops=1)
                     ->  Materialize  (cost=0.00..72.00 rows=2000 width=6) (actual time=0.061..110.777 rows=4000000 loops=2000)
                           ->  Seq Scan on supplier t2  (cost=0.00..62.00 rows=2000 width=6) (actual time=0.004..0.594 rows=2000 loops=1)
 Total runtime: 995.772 ms
(12 rows)
```

使用 MAX 聚集函数

实验结果

```
 s_suppkey |          s_name
-----------+---------------------------
       892 | Supplier#000000892
(1 row)
```

执行时间: 2.550 ms

```
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Seq Scan on supplier  (cost=67.01..134.01 rows=1 width=30) (actual time=1.894..2.413 rows=1 loops=1)
   Filter: (s_acctbal = $0)
   Rows Removed by Filter: 1999
   InitPlan 1 (returns $0)
     ->  Aggregate  (cost=67.00..67.01 rows=1 width=38) (actual time=1.416..1.416 rows=1 loops=1)
           ->  Seq Scan on supplier  (cost=0.00..62.00 rows=2000 width=6) (actual time=0.005..0.556 rows=2000 loops=1)
 Total runtime: 2.550 ms
(7 rows)
```

1.3.4 多表查询

查询7:

选取两张数据量比较小的表 `T1` 和 `T2` （如 `REGION`、`NATION`、`SUPPLIER`），执行如下无连接条件的笛卡尔积操作，观察数据库系统的反应和查询结果：

```
SELECT *
FROM REGION, NATION;
```

实验结果

```
r_regionkey |          r_name           |          r_comment           | n
_nationkey |          n_name           | n_regionkey |
                                  n_comment
------------+---------------------------+------------------------------+-
-----------+---------------------------+------------+------------------
--------------------------------------------------------------------------
----------------------
           0 | AFRICA                    | furiously special foxes hagg |
          0 | ALGERIA                   |           0 | posits use careful
ly pending accounts. special deposits haggle. ironic, silent accounts are
furio
           1 | AMERICA                   | furiously special foxes hagg |
          0 | ALGERIA                   |           0 | posits use careful
ly pending accounts. special deposits haggle. ironic, silent accounts are
furio
           2 | ASIA                      | furiously special foxes hagg |
          0 | ALGERIA                   |           0 | posits use careful
ly pending accounts. special deposits haggle. ironic, silent accounts are
furio
           3 | EUROPE                    | furiously special foxes hagg |
          0 | ALGERIA                   |           0 | posits use careful
ly pending accounts. special deposits haggle. ironic, silent accounts are
furio
           4 | MIDDLE EAST               | furiously special foxes hagg |
          0 | ALGERIA                   |           0 | posits use careful
ly pending accounts. special deposits haggle. ironic, silent accounts are
furio
           0 | AFRICA                    | furiously special foxes hagg |
          1 | ARGENTINA                 |           1 | ly bold instructio
ns haggle quickly across the blithely close dep
           1 | AMERICA                   | furiously special foxes hagg |
          1 | ARGENTINA                 |           1 | ly bold instructio
ns haggle quickly across the blithely close dep
           2 | ASIA                      | furiously special foxes hagg |
          1 | ARGENTINA                 |           1 | ly bold instructio
ns haggle quickly across the blithely close dep
           3 | EUROPE                    | furiously special foxes hagg |
          1 | ARGENTINA                 |           1 | ly bold instructio
ns haggle quickly across the blithely close dep
           4 | MIDDLE EAST               | furiously special foxes hagg |
          1 | ARGENTINA                 |           1 | ly bold instructio
ns haggle quickly across the blithely close dep
           .........
           3 | EUROPE                    | furiously special foxes hagg |
         10 | IRAN                      |           4 | equests. packages
are ironic, regular theodolites. carefully regular ideas sleep slyly final
, ex
```

```
15            4 | MIDDLE EAST               | furiously special foxes hagg |
          10 | IRAN                       |           4 | equests. packages
are ironic, regular theodolites. carefully regular ideas sleep slyly final
, ex
16            0 | AFRICA                    | furiously special foxes hagg |
          11 | IRAQ                       |           4 | cording to the qui
ckly regular platelets. carefully ironic pinto beans against the slyly unu
sual theodolites d
17            1 | AMERICA                   | furiously special foxes hagg |
          11 | IRAQ                       |           4 | cording to the qui
ckly regular platelets. carefully ironic pinto beans against the slyly unu
sual theodolites d
18            2 | ASIA                      | furiously special foxes hagg |
          11 | IRAQ                       |           4 | cording to the qui
ckly regular platelets. carefully ironic pinto beans against the slyly unu
sual theodolites d
19            3 | EUROPE                    | furiously special foxes hagg |
          11 | IRAQ                       |           4 | cording to the qui
ckly regular platelets. carefully ironic pinto beans against the slyly unu
sual theodolites d
20            ........
21 (125 row)
```

查询8:

使用多表连接操作，从订单表 `ORDERS` 、供应商表 `SUPPLIER` 、订单明细表 `LINEITEM` 中，查询实际到达日期小于预计到达日期的订单，列出这些订单的订单 `key` 、订单总价、下单日期以及该供应商的姓名、地址和手机号。

```sql
SELECT O.O_ORDERKEY, O.O_TOTALPRICE, O.O_ORDERDATE, S.S_NAME, S.S_ADDRESS,
S.S_PHONE
FROM ORDERS O
JOIN LINEITEM L ON O.O_ORDERKEY = L.L_ORDERKEY
JOIN SUPPLIER S ON L.L_SUPPKEY = S.S_SUPPKEY
WHERE L.L_RECEIPTDATE < L.L_COMMITDATE;
```

实验结果

```
 o_orderkey | o_totalprice |     o_orderdate     |          s_name
  | s_address  |     s_phone
------------+--------------+---------------------+------------------------
--+------------+-----------------
       59108 |    268538.27 | 2018-10-08 00:00:00 | Supplier#000001022
  | 0000000000 | 24-859-889-7512
       66787 |    233043.61 | 2015-12-12 00:00:00 | Supplier#000001512
  | 0000000000 | 33-670-389-3311
       85475 |    217043.30 | 2017-05-24 00:00:00 | Supplier#000000994
  | 0000000000 | 14-183-331-6019
       96772 |    255936.60 | 2017-03-04 00:00:00 | Supplier#000001303
  | 0000000000 | 22-688-457-2776
       98022 |    331558.89 | 2017-06-07 00:00:00 | Supplier#000001321
  | 0000000000 | 32-708-579-1992
       .............
(56424 row)
```

查询9：

使用多表连接操作，从供应商表 `SUPPLIER` 、零部件表 `PART` 、零部件供应表 `PARTSUPP` 中，查询供应零件品牌为 `'Brand#13'` 的供应商信息，列出零件供应数量与成本，以及供应商的姓名与手机号。

```sql
SELECT PS.PS_AVAILQTY, PS.PS_SUPPLYCOST, S.S_NAME, S.S_PHONE
FROM PARTSUPP PS
JOIN SUPPLIER S ON PS.PS_SUPPKEY = S.S_SUPPKEY
JOIN PART P ON PS.PS_PARTKEY = P.P_PARTKEY
WHERE P.P_BRAND = 'Brand#13';
```

实验结果

```
 ps_availqty | ps_supplycost |          s_name           |     s_phone
-------------+---------------+---------------------------+---------------
-
           1 |        771.64 | Supplier#000000002        | 15-679-861-2259
           1 |        993.49 | Supplier#000000502        | 14-678-262-5636
           1 |        337.09 | Supplier#000001002        | 32-102-374-6308
           1 |        357.84 | Supplier#000001502        | 12-226-454-8297
           1 |        378.49 | Supplier#000000003        | 11-383-516-1199
           1 |        915.27 | Supplier#000000503        | 30-263-152-1630
           1 |        438.37 | Supplier#000001003        | 20-763-167-9528
           .........
(6424 row)
```

查询10：

利用订单明细表 `LINEITEM`，使用元组变量方式，查询所有比流水号为 `'1'`，订单号为 `'1'` 的折扣高的订单 `key` 和流水号，列出这些订单的零件、折扣，结果按照折扣的降序排列。

```
SELECT L1.L_ORDERKEY, L1.L_LINENUMBER, L1.L_PARTKEY, L1.L_DISCOUNT
FROM LINEITEM L1
WHERE L1.L_DISCOUNT > (
    SELECT L2.L_DISCOUNT
    FROM LINEITEM L2
    WHERE L2.L_ORDERKEY = '1' AND L2.L_LINENUMBER = '1'
)
ORDER BY L1.L_DISCOUNT DESC;
```

实验结果

```
 l_orderkey | l_linenumber | l_partkey | l_discount
------------+--------------+-----------+------------
     940324 |            1 |     29535 |        .10
    1114981 |            1 |       914 |        .10
     831842 |            3 |     34597 |        .10
     172641 |            3 |      9552 |        .10
     411648 |            2 |      8455 |        .10
     179014 |            4 |     18310 |        .10
     683425 |            7 |     15672 |        .10
    1109477 |            4 |     31501 |        .10
    1109476 |            4 |      1137 |        .10
    .........
(493904 row)
```

1.3.5 聚集函数

查询11:

从订单明细表 `LINEITEM`、订单表 `ORDERS`、客户表 `CUSTOMER`、国家表 `NATION`，查询客户来自 `ALGERIA`，下单日期为 `'2015-01-01'` 到 `'2015-02-02'` 的订单下列信息:

(1) 满足条件订单的最大数量、最小数量和平均数量。

(2) 具有最大数量且满足上述条件的订单，列出该订单的发货日期、下单日期。

```sql
-- (1)
SELECT MAX(L.L_QUANTITY) AS MAX_QTY, MIN(L.L_QUANTITY) AS MIN_QTY, AVG(L.L_QUANTITY) AS AVG_QTY
FROM LINEITEM L
JOIN ORDERS O ON L.L_ORDERKEY = O.O_ORDERKEY
JOIN CUSTOMER C ON O.O_CUSTKEY = C.C_CUSTKEY
JOIN NATION N ON C.C_NATIONKEY = N.N_NATIONKEY
WHERE N.N_NAME = 'ALGERIA'
  AND O.O_ORDERDATE BETWEEN '2015-01-01'::DATE AND '2015-02-02'::DATE;

-- (2)
SELECT L.L_QUANTITY, L.L_SHIPDATE, O.O_ORDERDATE
FROM LINEITEM L
JOIN ORDERS O ON L.L_ORDERKEY = O.O_ORDERKEY
JOIN CUSTOMER C ON O.O_CUSTKEY = C.C_CUSTKEY
JOIN NATION N ON C.C_NATIONKEY = N.N_NATIONKEY
WHERE N.N_NAME = 'ALGERIA'
  AND O.O_ORDERDATE BETWEEN '2015-01-01'::DATE AND '2015-02-02'::DATE
ORDER BY L.L_QUANTITY DESC
LIMIT 1;
```

实验结果-- (1)

```
 max_qty | min_qty |       avg_qty
---------+---------+--------------------
   50.00 |    1.00 | 25.5653962492437992
(1 row)
```

实验结果-- (2)

```
 l_quantity |     l_shipdate      |     o_orderdate
------------+---------------------+---------------------
      50.00 | 2015-02-02 00:00:00 | 2015-01-05 00:00:00
(1 row)
```

查询12:

根据零部件表 `PART` 和零部件供应表 `PARTSUPP` 及供应商表 `SUPPLIER`，查询有多少零件厂商提供了品牌为 `Brand#13` 的零件，给出这些零件的类型、零售价和供应商数量，并将查询结果按照零售价降序排列。

```sql
SELECT P.P_TYPE, P.P_RETAILPRICE, COUNT(DISTINCT S.S_SUPPKEY) AS SUPPLIER_C
OUNT
FROM PART P
JOIN PARTSUPP PS ON P.P_PARTKEY = PS.PS_PARTKEY
JOIN SUPPLIER S ON PS.PS_SUPPKEY = S.S_SUPPKEY
WHERE P.P_BRAND = 'Brand#13'
GROUP BY P.P_TYPE, P.P_RETAILPRICE
ORDER BY P.P_RETAILPRICE DESC
```

实验结果

```
        p_type         | p_retailprice | supplier_count
-----------------------+---------------+----------------
 STANDARD ANODIZED TIN |       1932.99 |              8
 STANDARD ANODIZED TIN |       1930.99 |              4
 STANDARD ANODIZED TIN |       1929.99 |              4
 STANDARD ANODIZED TIN |       1923.99 |              8
 ............
 (1349 row)
```

查询13：

从零部件表 `PART` 和零部件供应表 `PARTSUPP` 中，查询所有零件大小在 `[7,14]` 之间的零件的平均零售价，给出零件 `key`，供应成本，平均零售价，结果按照零售价降序排列。

```sql
SELECT P.P_PARTKEY, PS.PS_SUPPLYCOST, AVG(P.P_RETAILPRICE) AS AVG_RETAILPRI
CE
FROM PART P
JOIN PARTSUPP PS ON P.P_PARTKEY = PS.PS_PARTKEY
WHERE P.P_SIZE BETWEEN 7 AND 14
GROUP BY P.P_PARTKEY, PS.PS_SUPPLYCOST
ORDER BY AVG_RETAILPRICE DESC;
```

实验结果

```
 p_partkey | ps_supplycost |    avg_retailprice
-----------+---------------+----------------------
     39998 |        254.68 | 1937.9900000000000000
     39998 |        711.23 | 1937.9900000000000000
     39998 |        755.19 | 1937.9900000000000000
     39998 |        880.04 | 1937.9900000000000000
     35999 |        215.76 | 1934.9900000000000000
     35999 |        154.96 | 1934.9900000000000000
 ............
     22998 |        455.36 | 1920.9900000000000000
     24996 |        698.30 | 1920.9900000000000000
     21999 |         71.50 | 1920.9900000000000000
     22998 |        916.71 | 1920.9900000000000000
     22998 |         11.62 | 1920.9900000000000000
     24996 |        152.65 | 1920.9900000000000000
 ............
(26084 row)
```

1.3.6 嵌套查询

查询14：

从订单明细表 `LINEITEM` 、订单表 `ORDERS` 、客户表 `CUSTOMER` 中，使用 `IN` 运算符，查询明细折扣小于 `0.01` 的订单，列出这些订单的 `key` 和采购订单的客户姓名。

对比使用多表连接、非嵌套的查询在执行时间、查询结果上的异同。

```sql
-- 使用嵌套查询
EXPLAIN ANALYZE
SELECT O.O_ORDERKEY, C.C_NAME
FROM ORDERS O
JOIN CUSTOMER C ON O.O_CUSTKEY = C.C_CUSTKEY
WHERE O.O_ORDERKEY IN (
    SELECT L.L_ORDERKEY
    FROM LINEITEM L
    WHERE L.L_DISCOUNT < 0.01
);

-- 使用多表连接
EXPLAIN ANALYZE
SELECT DISTINCT O.O_ORDERKEY, C.C_NAME
FROM ORDERS O
JOIN CUSTOMER C ON O.O_CUSTKEY = C.C_CUSTKEY
JOIN LINEITEM L ON O.O_ORDERKEY = L.L_ORDERKEY
WHERE L.L_DISCOUNT < 0.01;
```

使用嵌套查询

实验结果

```
 o_orderkey |       c_name
------------+--------------------
          2 | Customer#000015601
         34 | Customer#000012202
         65 | Customer#000003251
         71 | Customer#000000676
         98 | Customer#000020896
        100 | Customer#000029401
        101 | Customer#000005600
        133 | Customer#000008800
        .......
       2215 | Customer#000007738
       2241 | Customer#000020257
       2273 | Customer#000026851
       2304 | Customer#000008986
       2305 | Customer#000008395
       2306 | Customer#000005342
        .......
       4994 | Customer#000008488
       4995 | Customer#000007742
       4996 | Customer#000026572
       4997 | Customer#000009260
       5025 | Customer#000023927
       5027 | Customer#000029248
       5029 | Customer#000002077
        .......
```

执行时间: 403.720 ms

```
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=99251.34..108238.79 rows=62217 width=23) (actual time=328.351..401.525 rows=70400 loops=1)
   Hash Cond: (o.o_custkey = c.c_custkey)
   ->  Hash Join  (cost=97857.34..105989.30 rows=62217 width=8) (actual time=316.412..376.821 rows=70400 loops=1)
         Hash Cond: (o.o_orderkey = l.l_orderkey)
         ->  Seq Scan on orders o  (cost=0.00..7286.00 rows=300000 width=8) (actual time=0.004..25.497 rows=300001 loops=1)
         ->  Hash  (cost=97737.39..97737.39 rows=9596 width=4) (actual time=316.189..316.189 rows=70400 loops=1)
               Buckets: 32768  Batches: 1  Memory Usage: 2475kB
               ->  HashAggregate  (cost=97641.43..97737.39 rows=9596 width=4) (actual time=303.742..310.659 rows=70400 loops=1)
                     Group By Key: l.l_orderkey
                     ->  Seq Scan on lineitem l  (cost=0.00..97408.30 rows=93252 width=4) (actual time=0.063..287.357 rows=81498 loops=1)
                           Filter: (l_discount < .01)
                           Rows Removed by Filter: 822355
   ->  Hash  (cost=1019.00..1019.00 rows=30000 width=23) (actual time=11.747..11.747 rows=30000 loops=1)
         Buckets: 32768  Batches: 1  Memory Usage: 1641kB
         ->  Seq Scan on customer c  (cost=0.00..1019.00 rows=30000 width=23) (actual time=0.010..6.179 rows=30000 loops=1)
 Total runtime: 403.720 ms
(16 rows)
```

使用多表连接

实验结果

```
 o_orderkey |       c_name
------------+--------------------
     674469 | Customer#000002546
     835143 | Customer#000003364
     964549 | Customer#000026950
    1049190 | Customer#000003877
     223141 | Customer#000025421
     ........
      49056 | Customer#000013006
     526529 | Customer#000022004
     599013 | Customer#000022811
     255523 | Customer#000009337
     703137 | Customer#000007178
     822976 | Customer#000026330
     ........
     393604 | Customer#000012704
     289568 | Customer#000015125
     231591 | Customer#000023413
    1067269 | Customer#000004435
     680103 | Customer#000025159
```

执行时间: 452.057 ms

```
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=112868.99..113801.51 rows=93252 width=23) (actual time=440.946..448.941 rows=70400 loops=1)
   Group By Key: o.o_orderkey, c.c_name
   ->  Hash Join  (cost=12430.00..112402.73 rows=93252 width=23) (actual time=80.918..421.119 rows=81498 loops=1)
         Hash Cond: (o.o_custkey = c.c_custkey)
         ->  Hash Join  (cost=11036.00..109726.52 rows=93252 width=8) (actual time=69.279..390.006 rows=81498 loops=1)
               Hash Cond: (l.l_orderkey = o.o_orderkey)
               ->  Seq Scan on lineitem l  (cost=0.00..97408.30 rows=93252 width=4) (actual time=0.037..294.453 rows=81498 loops=1)
                     Filter: (l_discount < .01)
                     Rows Removed by Filter: 822355
               ->  Hash  (cost=7286.00..7286.00 rows=300000 width=8) (actual time=67.846..67.846 rows=300001 loops=1)
                     Buckets: 524288  Batches: 1  Memory Usage: 11719kB
                     ->  Seq Scan on orders o  (cost=0.00..7286.00 rows=300000 width=8) (actual time=0.008..37.125 rows=300001 loops=1)
         ->  Hash  (cost=1019.00..1019.00 rows=30000 width=23) (actual time=11.364..11.364 rows=30000 loops=1)
               Buckets: 32768  Batches: 1  Memory Usage: 1641kB
               ->  Seq Scan on customer c  (cost=0.00..1019.00 rows=30000 width=23) (actual time=0.011..6.277 rows=30000 loops=1)
 Total runtime: 452.057 ms
(16 rows)
```

查询15-1:

从订单明细表 `LINEITEM`，使用 `SOME` 运算符，查询满足下列条件的订单：该订单的数量大于发货日期在 `[开始日期 2018-10-10, 结束日期 2021-10-10]` 之间的部分（至少一个）订单的数量，列出这些订单的流水号、 `key` 和税。

```sql
SELECT L_ORDERKEY, L_LINENUMBER, L_TAX
FROM LINEITEM
WHERE L_QUANTITY > SOME (
    SELECT L_QUANTITY
    FROM LINEITEM
    WHERE L_SHIPDATE BETWEEN '2021-1-9'::DATE AND '2021-1-10'::DATE
);
```

实验结果

```
 l_orderkey | l_linenumber | l_tax
------------+--------------+-------
     126017 |            3 |   .03
     126017 |            4 |   .06
     126017 |            5 |  0.00
     126017 |            6 |   .05
     126018 |            1 |   .07
     126049 |            1 |  0.00
     126080 |            1 |   .02
     ........
     126884 |            4 |   .07
     126884 |            5 |   .05
     126885 |            1 |   .08
     126914 |            1 |  0.00
     126914 |            2 |   .01
     ........
     127524 |            1 |   .04
     127524 |            2 |   .02
     127524 |            3 |   .08
     127524 |            4 |   .03
     127524 |            5 |   .03
     ........
```

查询15-2：

从订单表 `ORDERS`，使用 `SOME` 运算符，查询满足下列条件的订单：订单状态为 `'O'`，订单总价大于部分在 `2020` 年之后下单的订单。列出这些订单的 `key`、客户 `key`、收银员。

```sql
SELECT O_ORDERKEY, O_CUSTKEY, O_CLERK
FROM ORDERS
WHERE O_ORDERSTATUS = 'O'
  AND O_TOTALPRICE > SOME (
    SELECT O_TOTALPRICE
    FROM ORDERS
    WHERE O_ORDERDATE >= '2020-01-01'::DATE
  );
SELECT count(*)
FROM ORDERS
WHERE O_ORDERSTATUS = 'O'
  AND O_TOTALPRICE > SOME (
    SELECT O_TOTALPRICE
    FROM ORDERS
    WHERE O_ORDERDATE >= '2020-01-01'::DATE
  );
```

实验结果

```
 o_orderkey | o_custkey |     o_clerk
------------+-----------+-----------------
          1 |      7381 | Clerk#000000951
          2 |     15601 | Clerk#000000880
          4 |     27356 | Clerk#000000124
          7 |      7828 | Clerk#000000470
         32 |     26012 | Clerk#000000616
         34 |     12202 | Clerk#000000223
        .......
(146319 row)
```

查询16-1：

从订单明细表 `LINEITEM` 中，使用 `>= ALL` 运算符，查询满足下列条件的供应商：该供应商在 `2019` 年出货量大于等于同时段其他供应商的出货量，即 `2019` 年该供应商的出货量最高。

```sql
SELECT L.L_SUPPKEY
FROM LINEITEM L
WHERE L.L_SHIPDATE BETWEEN '2019-01-01'::DATE AND '2019-12-31'::DATE
GROUP BY L.L_SUPPKEY
HAVING SUM(L.L_QUANTITY) >= ALL (
    SELECT SUM(L2.L_QUANTITY)
    FROM LINEITEM L2
    WHERE L2.L_SHIPDATE BETWEEN '2019-01-01'::DATE AND '2019-12-31'::DATE
    GROUP BY L2.L_SUPPKEY
);
```

实验结果

```
 l_suppkey
-----------
       370
(1 row)
```

查询16-2：

供应商表 `SUPPLIER`，使用 `ALL` 运算符，查询账户余额大于等于其他供应商的供应商。列出该供应商的姓名、`key`、手机号。

```sql
SELECT S_SUPPKEY, S_NAME, S_PHONE
FROM SUPPLIER
WHERE S_ACCTBAL >= ALL (
    SELECT S_ACCTBAL
    FROM SUPPLIER
);
```

实验结果

```
 s_suppkey |          s_name           |     s_phone
-----------+---------------------------+-----------------
       892 | Supplier#000000892        | 18-893-665-3629
(1 row)
```

查询17-1:

从供应商表 `SUPPLIER`、国家表 `NATION`，使用 `EXISTS` 运算符，查询国家为日本，账户余额大于 `5000` 的供应商。

```sql
SELECT S.S_SUPPKEY, S.S_NAME, S.S_ACCTBAL
FROM SUPPLIER S
WHERE S.S_NATIONKEY = (
    SELECT N.N_NATIONKEY
    FROM NATION N
    WHERE N.N_NAME = 'JAPAN'
)
  AND S.S_ACCTBAL > 5000;
```

实验结果

```
 s_suppkey |          s_name           | s_acctbal
-----------+---------------------------+-----------
        43 | Supplier#000000043        |   7773.41
       143 | Supplier#000000143        |   9658.99
       163 | Supplier#000000163        |   7999.27
       173 | Supplier#000000173        |   9583.11
       175 | Supplier#000000175        |   9845.98
       215 | Supplier#000000215        |   6125.89
       .......
      1568 | Supplier#000001568        |   7834.92
      1570 | Supplier#000001570        |   7963.33
      1614 | Supplier#000001614        |   9896.02
      1631 | Supplier#000001631        |   7687.91
      1638 | Supplier#000001638        |   8611.17
      1661 | Supplier#000001661        |   6817.13
      1681 | Supplier#000001681        |   6144.37
      1741 | Supplier#000001741        |   5050.43
      1862 | Supplier#000001862        |   6697.54
      1875 | Supplier#000001875        |   9358.58
      1886 | Supplier#000001886        |   6449.94
(42 rows)

```

查询17-2：

从客户表 `CUSTOMER`、国家表 `NATION`、订单表 `ORDERS`、订单明细表 `LINEITEM`、供应商表 `SUPPLIER` 中，使用 `NOT EXISTS EXCEPT` 运算符，查询满足下列条件的供应商：该供应商不能供应所有的零件。

```
SELECT S.S_SUPPKEY, S.S_NAME
FROM SUPPLIER S
WHERE NOT EXISTS (
    SELECT P.P_PARTKEY
    FROM PART P
    EXCEPT
    SELECT PS.PS_PARTKEY
    FROM PARTSUPP PS
    WHERE PS.PS_SUPPKEY = S.S_SUPPKEY
);
```

实验结果

```
 n_nationkey |          n_name
-------------+---------------------------
           0 | ALGERIA
(1 row)
```

查询18：

从国家表 `NATION`、客户表 `CUSTOMER` 中，使用 `COUNT`，查询满足下列条件的国家：至少有 `3` 个客户来自这个国家，并列出该国家的国家 `key` 和国家名。

```
SELECT N.N_NATIONKEY, N.N_NAME
FROM NATION N
JOIN CUSTOMER C ON N.N_NATIONKEY = C.C_NATIONKEY
GROUP BY N.N_NATIONKEY, N.N_NAME
HAVING COUNT(C.C_CUSTKEY) >= 3;
```

实验结果

```
 n_nationkey |          n_name
-------------+---------------------------
           0 | ALGERIA
(1 row)
```

查询19：

从零部件表 `PART` 和零部件供应表 `PARTSUPP` 中，使用 `FROM` 子句中的子查询，查询满足下列条件的零件：零件由 `2` 个以上的供应商供应，且零件大小在 `20` 以上。

```sql
SELECT T.PS_PARTKEY
FROM (
    SELECT PS.PS_PARTKEY, P.P_SIZE, COUNT(DISTINCT PS.PS_SUPPKEY) AS SUPP_COUNT
    FROM PART P
    JOIN PARTSUPP PS ON P.P_PARTKEY = PS.PS_PARTKEY
    GROUP BY PS.PS_PARTKEY, P.P_SIZE
    HAVING COUNT(DISTINCT PS.PS_SUPPKEY) > 2
) T
WHERE T.P_SIZE >= 20;
```

实验结果

```
 ps_partkey
------------
           3
           7
           8
          10
          11
          12
.......
       24593
```

1.3.7 WITH 临时视图查询

查询20：

用 `WITH` 临时视图方式，实现查询19中的查询要求。

```sql
WITH TEMP AS (
    SELECT PS.PS_PARTKEY, P.P_SIZE, COUNT(DISTINCT PS.PS_SUPPKEY) AS SUPP_COUNT
    FROM PART P
    JOIN PARTSUPP PS ON P.P_PARTKEY = PS.PS_PARTKEY
    GROUP BY PS.PS_PARTKEY, P.P_SIZE
    HAVING COUNT(DISTINCT PS.PS_SUPPKEY) > 2
)
SELECT T.PS_PARTKEY
FROM TEMP T
WHERE T.P_SIZE >= 20;
```

实验结果

```
ps_partkey
------------
           3
           7
           8
          10
          11
          12
          14
.......
         211
         212
         214
         216
         217
         218
         219
.......
         24593
```

查询21:

从零部件供应表 `PARTSUPP` 中，用 `WITH` 临时视图方式，查询零件供应数量最多的供应商 `key` 和其供应的数量。

```sql
WITH SUP_MAX AS (
    SELECT PS.PS_SUPPKEY, SUM(PS.PS_AVAILQTY) AS TOTAL_QTY
    FROM PARTSUPP PS
    GROUP BY PS.PS_SUPPKEY
)
SELECT S.PS_SUPPKEY, S.TOTAL_QTY
FROM SUP_MAX S
WHERE S.TOTAL_QTY = (
    SELECT MAX(TOTAL_QTY)
    FROM SUP_MAX
);
```

实验结果

```
 ps_suppkey | total_qty
------------+-----------
          33 |        81
        1033 |        81
         533 |        81
        1533 |        81
(4 rows)
```

1.3.8 键/函数依赖分析

查询22：

在订单明细表 `LINEITEM` 中，检查订单 `key` 、零件 `key` 、供应商 `key` 、流水号是否组成超键。

```
SELECT L_ORDERKEY, L_PARTKEY, L_SUPPKEY, L_LINENUMBER, COUNT(*) AS COUNT
FROM LINEITEM
GROUP BY L_ORDERKEY, L_PARTKEY, L_SUPPKEY, L_LINENUMBER
HAVING COUNT(*) > 1;
```

如果查询结果为空，说明上述字段组合能唯一标识一条记录，组成超键。

实验结果

```
 l_orderkey | l_partkey | l_suppkey | l_linenumber | count
------------+-----------+-----------+--------------+-------
(0 rows)
```

查询23：

在订单明细表 `LINEITEM` 中，利用 SQL 语句检查函数依赖 `L_PARTKEY → L_EXTENDEDPRICE` 是否成立；如果不成立，利用 SQL 语句找出导致函数依赖不成立的元组。

```
-- 检查函数依赖是否成立
SELECT L_PARTKEY
FROM LINEITEM
GROUP BY L_PARTKEY
HAVING COUNT(DISTINCT L_EXTENDEDPRICE) > 1;

-- 找出导致函数依赖不成立的元组(由于元组过长，使用三个属性代表元组)
SELECT All l_orderkey,l_partkey,l_suppkey
FROM LINEITEM
WHERE L_PARTKEY IN (
    SELECT L_PARTKEY
    FROM LINEITEM
    GROUP BY L_PARTKEY
    HAVING COUNT(DISTINCT L_EXTENDEDPRICE) > 1
);
```

实验结果-- 检查函数依赖是否成立

```
 l_partkey
-----------
         1
         2
         3
         4
         5
         6
         7
         8
         9
        10
        .....
        21
        22
        23
        24
        25
        26
        27
        28
        .....
```

实验结果-- 找出导致函数依赖不成立的元组

```
 l_orderkey | l_partkey | l_suppkey
------------+-----------+-----------
        896 |     15262 |       784
        967 |     14400 |       908
       1413 |      5129 |       636
       2628 |       535 |        36
       2693 |     11550 |      1551
       4419 |     31124 |      1640
       5282 |       529 |        30
      10279 |     12702 |       703
      10532 |      5928 |       933
.......
     114339 |     17364 |       381
     118784 |     17674 |      1199
     121123 |     32420 |       421
     121221 |     31825 |      1826
     122021 |      3647 |       650
     123781 |      6795 |       796
     124804 |      5180 |      1683
     124839 |     11660 |      1661
.......
     251875 |     19522 |      1523
     252865 |     25721 |       746
     254117 |      5266 |      1769
     254177 |     28791 |       334
     255299 |     34388 |       389
     255552 |     20069 |      1090
.......
```

1.3.9 关系表的插入/删除/更新

查询24:

向订单表 `ORDERS` 中插入一条订单数据。

```sql
-- 插入新订单
INSERT INTO ORDERS (O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE, O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY, O_COMMENT)
VALUES
    ('1200001',      -- 订单号
     '20045',        -- 客户号
     'F',            -- 订单状态
     61365.24,       -- 订单总价
     '2017-03-19'::DATE, -- 下单日期
     '2-HIGH',       -- 订单优先级
     'Clerk#000000098', -- 收银员
     0,              -- 发货优先级
     'furiously special f'); -- 订单备注
```

实验结果

```
INSERT 0 1
```

查询25:

将零件 `32` 的全部供应商，作为零件 `20` 的供应商，加入到零部件供应表 `PARTSUPP` 中。

```sql
INSERT INTO PARTSUPP (PS_PARTKEY, PS_SUPPKEY, PS_AVAILQTY, PS_SUPPLYCOST, PS_COMMENT)
SELECT 20, PS_SUPPKEY, PS_AVAILQTY, PS_SUPPLYCOST, PS_COMMENT
FROM PARTSUPP
WHERE PS_PARTKEY = 32
  AND PS_SUPPKEY NOT IN (
    SELECT PS_SUPPKEY
    FROM PARTSUPP
    WHERE PS_PARTKEY = 20
  );
```

实验结果

```
INSERT 0 0
```

查询26:

在订单明细表 `LINEITEM` 中，删除已退货的订单记录（`L_RETURNFLAG = 'R'`）。

```sql
DELETE FROM LINEITEM
WHERE L_RETURNFLAG = 'R';
```

实验结果

```
DELETE 296116
```

查询27:

用订单明细表 `LINEITEM` 中在 `2019` 年之后交易中的预计到达日期，替换表中的实际到达日期。

```
UPDATE LINEITEM L
SET L_RECEIPTDATE = L_COMMITDATE
FROM ORDERS O
WHERE L.L_ORDERKEY = O.O_ORDERKEY
  AND O.O_ORDERDATE >= '2019-01-01'::DATE;
```

实验结果

```
UPDATE 470931
```

查询28:

针对订单明细表 `LINEITEM` 、订单表 `ORDERS` ，使用 `UPDATE` / `CASE` 语句做出如下修改：如果订单的订单优先级低于 `MEDIUM` ，则其在订单明细表中的预计到达日期推后 `2` 天，否则推迟一天。

```
UPDATE LINEITEM L
SET L_COMMITDATE = L_COMMITDATE + INTERVAL '1 day' * (
    CASE
        WHEN O.O_ORDERPRIORITY < '3-MEDIUM' THEN 2
        ELSE 1
    END
)
FROM ORDERS O
WHERE L.L_ORDERKEY = O.O_ORDERKEY;
```

实验结果

```
UPDATE 1199969
```

查询29:

在订单表 `ORDERS` 中，利用 `RANK` 函数，按照订单总价对订单进行降序排序，并输出订单 `key` 和排名。

```
SELECT O_ORDERKEY, O_TOTALPRICE, RANK() OVER (ORDER BY O_TOTALPRICE DESC) A
S "Rank"
FROM ORDERS;
```

实验结果

```
 o_orderkey | o_totalprice |  Rank
------------+--------------+--------
     209028 |    505770.15 |      1
     528388 |    497758.84 |      2
     993697 |    487758.42 |      3
    1111238 |    485577.76 |      4
     489319 |    484671.66 |      5
     366692 |    483521.14 |      6
     546785 |    481047.81 |      7
     326117 |    473020.26 |      8
     149509 |    471154.02 |      9
     185124 |    460604.60 |     10
     .......
    1024160 |    408809.93 |    193
      89859 |    408472.42 |    194
     135046 |    408452.16 |    195
     294343 |    408342.63 |    196
     885252 |    408223.13 |    197
     651718 |    408173.93 |    198
     189509 |    408153.63 |    199
     809125 |    408116.32 |    200
     .......
```