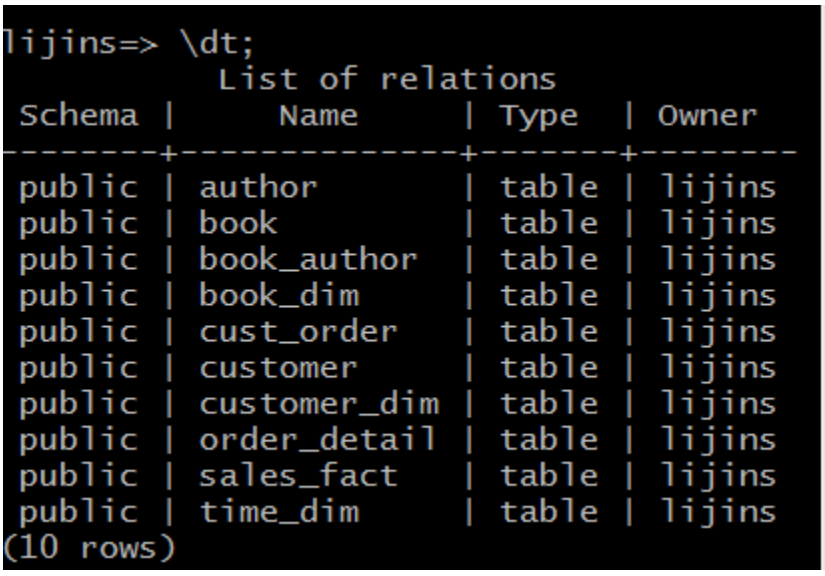
Q1.

Step 1:Create dimentsion and fact tables.

```
CREATE TABLE customer_dim (
   customerid integer primary key,
   l_name character(20) NOT NULL,
   f_name character(20),
   city character(15) NOT NULL,
   district character(15) NOT NULL,
   country character(15) NOT NULL,
   CONSTRAINT customer_customerid CHECK ((customerid > 0))
);

CREATE Table time_dim(
        timeid SERIAL primary key,
        orderdate date not null,
        dayofweek char(10) not null,
        month char(10) not null,
        year int not null
);

CREATE TABLE book_dim (
   isbn integer primary key,
   title character(60) NOT NULL,
   edition_no smallint DEFAULT 1,
   price numeric(6,2) NOT NULL,
   CONSTRAINT book_edition_no CHECK ((edition_no > 0)),
   CONSTRAINT book_price CHECK ((price > (0)::numeric))
);
```

```
CREATE table sales_fact(
        customerid int NOT NULL,
        timeid int NOT NULL,
   isbn int NOT NULL,
   amnt numeric(6,2) NOT NULL,
        FOREIGN KEY (customerid) REFERENCES customer_dim(customerid),
        FOREIGN KEY (timeid) REFERENCES time_dim(timeid),
 FOREIGN KEY (isbn) REFERENCES book_dim(isbn),
 CONSTRAINT amnt CHECK ((amnt > (0)::numeric))
);

ALTER TABLE ONLY sales_fact
   ADD CONSTRAINT sales_fact_pkey PRIMARY KEY (customerid, timeid, isbn);
```

Table list in DB:



```
lijins=> \dt;
          List of relations
 Schema |     Name      | Type  | Owner
--------+---------------+-------+--------
 public | author        | table | lijins
 public | book          | table | lijins
 public | book_author   | table | lijins
 public | book_dim      | table | lijins
 public | cust_order    | table | lijins
 public | customer      | table | lijins
 public | customer_dim  | table | lijins
 public | order_detail  | table | lijins
 public | sales_fact    | table | lijins
 public | time_dim      | table | lijins
(10 rows)
```

- book_dim is book dimension
- customer_dim is customer dimension
- time_dim is time dimension
- sales_fact is sales fact table

Step 2.Extract data from existing tables

1) customer dimension

```
INSERT INTO customer_dim (customerid, l_name, f_name, city, district, country)
 ( SELECT customerid,
        l_name,
        f_name,
        city,
        district,
        country
  FROM customer);
```

```
lijins=>INSERT 0 118
```

2) time dimension

```
INSERT INTO time_dim (orderdate, dayofweek, MONTH, YEAR)
 ( SELECT DISTINCT orderdate,
                to_char(orderdate,'Day'),
                to_char(orderdate,'Month'),
                to_number(to_char(orderdate,'YYYY'),'9999')
  FROM cust_order
  ORDER BY orderdate);
```

```
lijins=>INSERT 0 124
```

3) book dimension

```
INSERT INTO book_dim (isbn, title, edition_no, price)
 ( SELECT isbn, title, edition_no, price FROM book);
```

```
lijins=>INSERT 0 12
```

3) sales fact table

```
INSERT INTO sales_fact (customerid, timeid, isbn, amnt)
 ( SELECT cust_order_book_detail.customerid,
        time_dim.timeid,
        cust_order_book_detail.isbn,
```

```
        cust_order_book_detail.quantity * cust_order_book_detail.price AS amnt
 FROM
  (SELECT cust_order_detail.customerid,
         cust_order_detail.orderdate,
         cust_order_detail.isbn,
         cust_order_detail.quantity,
         book_dim.price
    FROM
     (SELECT cust_order.customerid,
           cust_order.orderdate,
           order_detail.isbn,
           sum(order_detail.quantity) quantity
      FROM cust_order
      NATURAL JOIN order_detail
      GROUP BY cust_order.customerid,
            cust_order.orderdate,
            order_detail.isbn) AS cust_order_detail
    NATURAL JOIN book_dim) AS cust_order_book_detail
 INNER JOIN time_dim ON cust_order_book_detail.orderdate = time_dim.orderdate);
```

```
lijins=>INSERT 0 1070
```

Q2.

create materialized view avg_amnt_view as

select customerid, avg(amnt) as avg_amnt from sales_fact

group by customerid;

```
lijins=>SELECT 104
```

Query1: select avg(avg_amnt) from avg_amnt_view;

```
lijins=>  select avg(avg_amnt) from avg_amnt_v
         avg
---------------------
 202.9588687852809865
(1 row)
```

Query2: select avg(amnt) from sales_fact;

```
lijins=> select avg(amnt) from sales_fact;
         avg
----------------------
 161.3691588785046729
(1 row)
```

The result of query1 is the average money spent per customer.

The result of query2 is the average money spent per customer, per orderdate, per book.isbn.

So the result of query2 is correct.

Q3.

a) Using "Roll-up" and "Dice" operations.

```
SELECT customer_dim.customerid,
      customer_dim.l_name,
      customer_dim.f_name
FROM customer_dim
NATURAL JOIN
  (SELECT customerid,
         sum(amnt) AS sum_amnt,
         rank() over (ORDER BY sum(amnt) DESC) rank
   FROM sales_fact
   GROUP BY customerid) AS tmp_view
WHERE tmp_view.rank < 6;
```

```
 customerid |         l_name         |         f_name
------------+------------------------+------------------------
          1 | Jacson                 | Kirk
          2 | Leow                   | May-N
          3 | Andree                 | Peter
         14 | Anslow                 | Craig
         79 | Liang                  | Jiajun
(5 rows)
```

b) Using "Slice" and "Roll-up" operation.

```
CREATE OR REPLACE FUNCTION best_perc_of_ord() RETURNS numeric AS $$
DECLARE
sum_amnt numeric;
ord_count int;
avg_ord_amnt numeric;
per_of_ord numeric;
greater_count int;
no_of_ord int;
BEGIN
 --get customer rank by amnt
 drop view if exists cust_sum_amnt_count_view;
```

```
create view cust_sum_amnt_count_view as
select customerid, sum(amnt) as sum_amnt, rank() over (order by sum(amnt) desc) rank, c
ount(1) from sales_fact
group by customerid;
--caculate amount of per order, per customer
drop view if exists cust_ord_amnt_view;
CREATE VIEW cust_ord_amnt_view as
 SELECT cust_order.customerid,
       ord_book_amnt.orderid,
       SUM(ord_book_amnt.amnt) sum_amnt
 FROM   cust_order
       NATURAL join (SELECT order_detail.orderid,
                     order_detail.isbn,
                     SUM(order_detail.quantity) * book_dim.price AS
                     amnt
                FROM   order_detail
                     NATURAL join book_dim
                GROUP  BY order_detail.orderid,
                       order_detail.isbn,
                       book_dim.price) AS ord_book_amnt
 GROUP  BY cust_order.customerid,
         ord_book_amnt.orderid;
sum_amnt = (SELECT sum(amnt) AS sum_amnt FROM sales_fact);
ord_count = (SELECT count(1) AS ord_count FROM cust_order);
avg_ord_amnt = sum_amnt/ord_count;
--debug: raise exception 'avg_ord_amnt: %', avg_ord_amnt;
--caculate count of by the best buyer orders whose amount greater than avg_ord_amnt
greater_count = (select count(1)
from cust_ord_amnt_view inner join cust_sum_amnt_count_view
on cust_ord_amnt_view.customerid = cust_sum_amnt_count_view.customerid
where cust_sum_amnt_count_view.rank = 1
and cust_ord_amnt_view.sum_amnt > avg_ord_amnt);
-- debug: raise exception 'greater_count: %', greater_count;
```

```
 -- caculate count of order by the best buyer
 no_of_ord = (select count(1) from cust_order natural join cust_sum_amnt_count_view
 where cust_sum_amnt_count_view.rank = 1);
 --debug::raise exception 'no_of_ord: %', no_of_ord;
 -- caculate count of order by best costomer
 per_of_ord = greater_count/no_of_ord :: numeric;
 -- debug::raise exception 'per_of_ord: %', per_of_ord;
 RETURN per_of_ord;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT best_perc_of_ord();
```

```
    best_perc_of_ord
------------------------
 0.71428571428571428571
(1 row)
```

Because best_perc_of_ord is 71.4% which is between 50% and 75% , we estimate that "the best buyer has issued a greater (than average) to medium number of orders with greater (than average) amounts of money.

Q4)

```
-- Create view1
DROP materialized VIEW IF EXISTS View1 CASCADE;
CREATE MATERIALIZED VIEW View1 AS
SELECT c.CustomerId, F_Name, L_Name, District, TimeId,
DayOfWeek, ISBN, Amnt
FROM Sales_fact NATURAL JOIN Customer_dim c NATURAL JOIN Time_dim;

-- Create view2
DROP materialized VIEW IF EXISTS View2 CASCADE;
CREATE MATERIALIZED VIEW View2 AS
SELECT c.CustomerId, F_Name, L_Name, Year, SUM(Amnt)
FROM Sales_fact NATURAL JOIN Customer_dim c NATURAL JOIN Time_dim
GROUP BY c.CustomerId, F_Name, L_Name, Year;

-- Create view3
DROP materialized VIEW IF EXISTS View3 CASCADE;
CREATE MATERIALIZED VIEW View3 AS
SELECT District, TimeId, DayOfWeek, ISBN, SUM(Amnt)
FROM Sales_fact NATURAL JOIN Customer_dim NATURAL JOIN Time_Dim
GROUP BY District, TimeId, DayOfWeek, ISBN;
```

a)

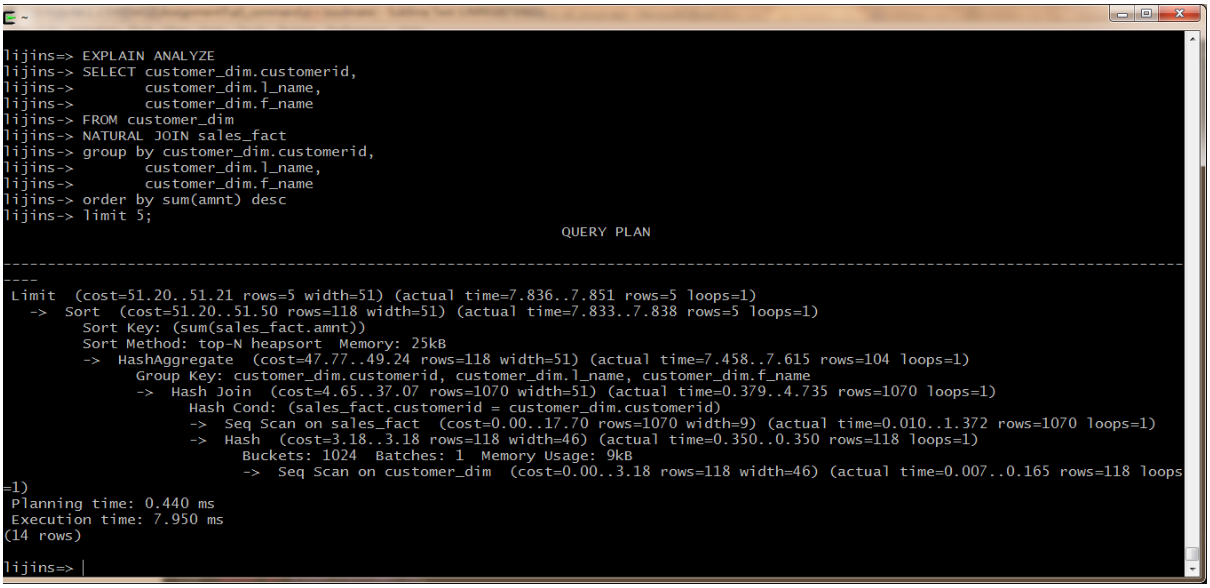
--1. The “Book Orders Database”

```
EXPLAIN ANALYZE
SELECT c.customerid,
    sum(b.price*od.quantity)
FROM customer c
NATURAL JOIN cust_order co
NATURAL JOIN order_detail od
NATURAL JOIN book b
GROUP BY c.customerid
ORDER BY sum(b.price*od.quantity) DESC
LIMIT 1;
```

```
                                                   QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------
------------------
 Limit  (cost=78.29..78.29 rows=1 width=20) (actual time=12.287..12.289 rows=1 loops=1)
   ->  Sort  (cost=78.29..78.58 rows=118 width=20) (actual time=12.282..12.282 rows=1 loops=1)
         Sort Key: (sum((b.price * (od.quantity)::numeric)))
         Sort Method: top-N heapsort  Memory: 25kB
         ->  HashAggregate  (cost=76.22..77.70 rows=118 width=20) (actual time=11.940..12.092 rows=104 loops=1)
               Group Key: c.customerid
               ->  Hash Join  (cost=15.97..65.22 rows=1100 width=20) (actual time=1.805..9.255 rows=1100 loops=1)
                     Hash Cond: (od.isbn = b.isbn)
                     ->  Hash Join  (cost=14.70..48.83 rows=1100 width=10) (actual time=1.689..6.189 rows=1100 loops=1)
                           Hash Cond: (od.orderid = co.orderid)
                           ->  Seq Scan on order_detail od  (cost=0.00..19.00 rows=1100 width=10) (actual time=0.008..1.426 rows=1100
loops=1)
                           ->  Hash  (cost=11.93..11.93 rows=222 width=8) (actual time=1.665..1.665 rows=222 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                 ->  Hash Join  (cost=4.65..11.93 rows=222 width=8) (actual time=0.426..1.311 rows=222 loops=1)
                                       Hash Cond: (co.customerid = c.customerid)
                                       ->  Seq Scan on cust_order co  (cost=0.00..4.22 rows=222 width=8) (actual time=0.009..0.280 row
s=222 loops=1)
                                       ->  Hash  (cost=3.18..3.18 rows=118 width=4) (actual time=0.355..0.355 rows=118 loops=1)
                                             Buckets: 1024  Batches: 1  Memory Usage: 5kB
                                             ->  Seq Scan on customer c  (cost=0.00..3.18 rows=118 width=4) (actual time=0.007..0.165
rows=118 loops=1)
                     ->  Hash  (cost=1.12..1.12 rows=12 width=18) (actual time=0.050..0.050 rows=12 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 1kB
                           ->  Seq Scan on book b  (cost=0.00..1.12 rows=12 width=18) (actual time=0.009..0.024 rows=12 loops=1)
 Planning time: 1.246 ms
 Execution time: 12.412 ms
(24 rows)
```

--2. The “The Data Mart”

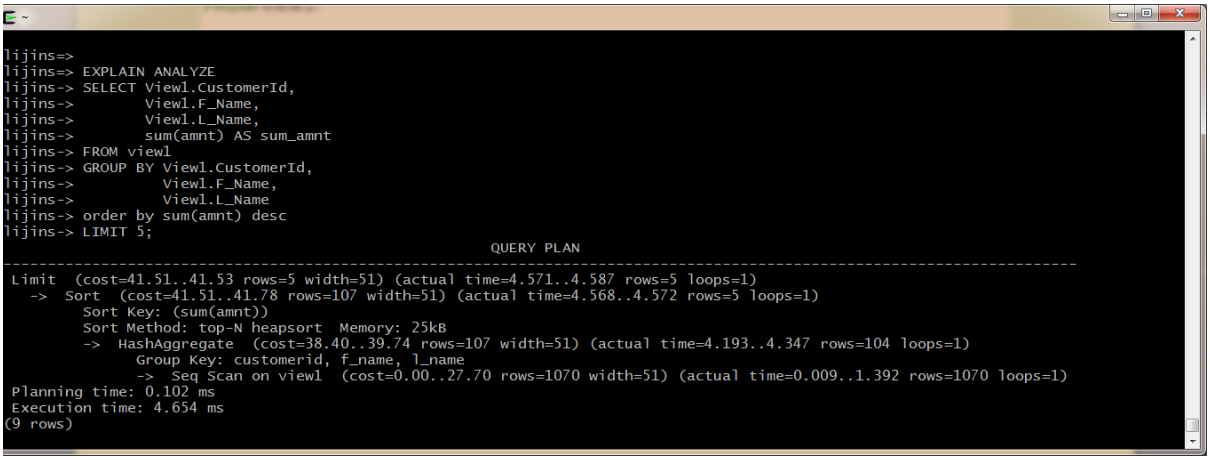
```
EXPLAIN ANALYZE
SELECT customer_dim.customerid,
        customer_dim.l_name,
        customer_dim.f_name
FROM customer_dim
NATURAL JOIN sales_fact
GROUP BY customer_dim.customerid,
            customer_dim.l_name,
            customer_dim.f_name
ORDER BY sum(amnt) DESC
LIMIT 5;
```



```
lijins=> EXPLAIN ANALYZE
lijins-> SELECT customer_dim.customerid,
lijins->        customer_dim.l_name,
lijins->        customer_dim.f_name
lijins-> FROM customer_dim
lijins-> NATURAL JOIN sales_fact
lijins-> group by customer_dim.customerid,
lijins->        customer_dim.l_name,
lijins->        customer_dim.f_name
lijins-> order by sum(amnt) desc
lijins-> limit 5;
                                                               QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------
----
 Limit  (cost=51.20..51.21 rows=5 width=51) (actual time=7.836..7.851 rows=5 loops=1)
   ->  Sort  (cost=51.20..51.50 rows=118 width=51) (actual time=7.833..7.838 rows=5 loops=1)
         Sort Key: (sum(sales_fact.amnt))
         Sort Method: top-N heapsort  Memory: 25kB
         ->  HashAggregate  (cost=47.77..49.24 rows=118 width=51) (actual time=7.458..7.615 rows=104 loops=1)
               Group Key: customer_dim.customerid, customer_dim.l_name, customer_dim.f_name
               ->  Hash Join  (cost=4.65..37.07 rows=1070 width=51) (actual time=0.379..4.735 rows=1070 loops=1)
                     Hash Cond: (sales_fact.customerid = customer_dim.customerid)
                     ->  Seq Scan on sales_fact  (cost=0.00..17.70 rows=1070 width=9) (actual time=0.010..1.372 rows=1070 loops=1)
                     ->  Hash  (cost=3.18..3.18 rows=118 width=46) (actual time=0.350..0.350 rows=118 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 9kB
                           ->  Seq Scan on customer_dim  (cost=0.00..3.18 rows=118 width=46) (actual time=0.007..0.165 rows=118 loops
=1)
 Planning time: 0.440 ms
 Execution time: 7.950 ms
(14 rows)

lijins=> |
```

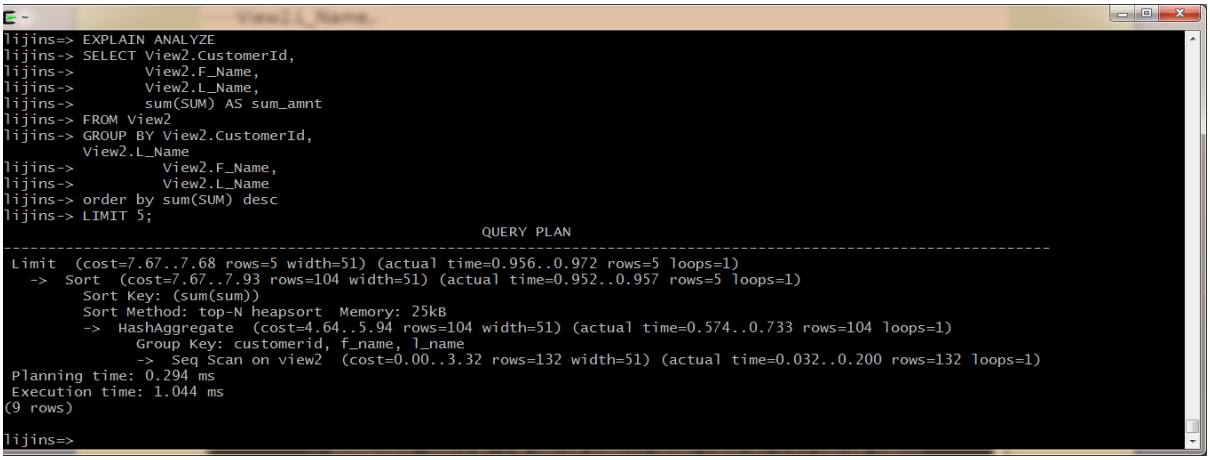
--3. The view View1

```
EXPLAIN ANALYZE
SELECT View1.CustomerId,
    View1.F_Name,
    View1.L_Name,
    sum(amnt) AS sum_amnt
FROM view1
GROUP BY View1.CustomerId,
     View1.F_Name,
     View1.L_Name
ORDER BY sum(amnt) DESC
LIMIT 5;
```



--4. The view View2

```
EXPLAIN ANALYZE
SELECT View2.CustomerId,
    View2.F_Name,
    View2.L_Name,
    sum(SUM) AS sum_amnt
FROM View2
GROUP BY View2.CustomerId,
     View2.F_Name,
     View2.L_Name
ORDER BY sum(SUM) DESC
LIMIT 5;
```



```
lijins=> EXPLAIN ANALYZE
lijins-> SELECT View2.CustomerId,
lijins->        View2.F_Name,
lijins->        View2.L_Name,
lijins->        sum(SUM) AS sum_amnt
lijins-> FROM View2
lijins-> GROUP BY View2.CustomerId,
        View2.L_Name
lijins->          View2.F_Name,
lijins->          View2.L_Name
lijins-> order by sum(SUM) desc
lijins-> LIMIT 5;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=7.67..7.68 rows=5 width=51) (actual time=0.956..0.972 rows=5 loops=1)
   ->  Sort  (cost=7.67..7.93 rows=104 width=51) (actual time=0.952..0.957 rows=5 loops=1)
         Sort Key: (sum(sum))
         Sort Method: top-N heapsort  Memory: 25kB
         ->  HashAggregate  (cost=4.64..5.94 rows=104 width=51) (actual time=0.574..0.733 rows=104 loops=1)
               Group Key: customerid, f_name, l_name
               ->  Seq Scan on view2  (cost=0.00..3.32 rows=132 width=51) (actual time=0.032..0.200 rows=132 loops=1)
 Planning time: 0.294 ms
 Execution time: 1.044 ms
(9 rows)

lijins=>
```

Explain the findings:

1) The speed of the execution of the query "2. The Data Mata" is faster than "1. The Book Orders Database". The reason is that the sales table already contains customerid and amnt information, which reduces the join time.

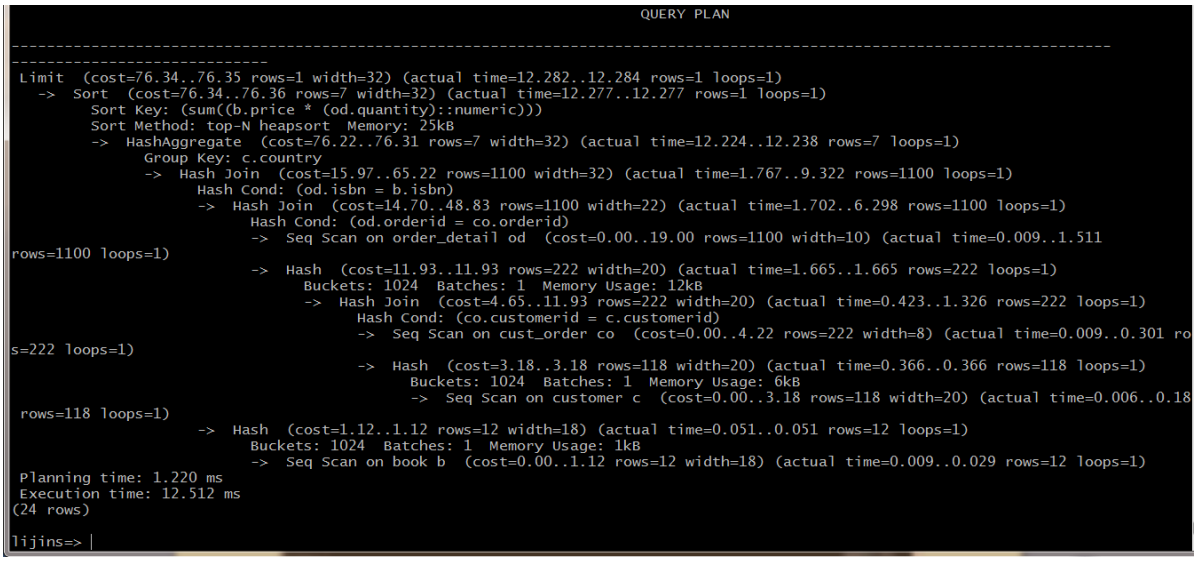
2) The speed of the execution of the query "3.The view View1" is faster than "2. The Data Mata ". Because materialized views improve query performance by pre-calculating expensive join and aggregation operations.

3) The speed of the execution of the query "4.The view View2" is faster than "3. The view View1 ". Because View1 only includes join operations, whereas, View2 includes join and aggregation operations.

Q4-b)

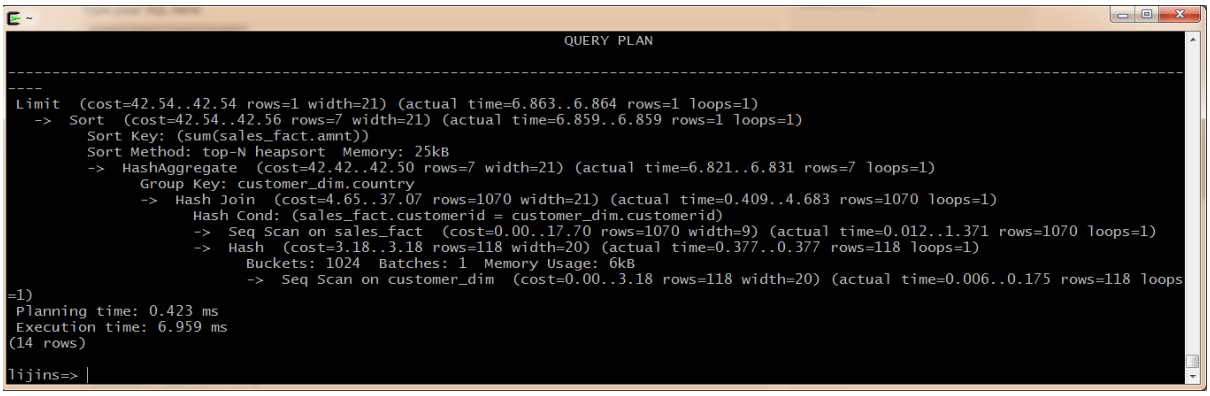
--1. The “Book Orders Database”

```
EXPLAIN ANALYZE
SELECT c.country, sum(b.price*od.quantity)
FROM customer c
NATURAL JOIN cust_order co
NATURAL JOIN order_detail od
NATURAL JOIN book b
GROUP BY c.country
ORDER BY sum(b.price*od.quantity) DESC
LIMIT 1;
```



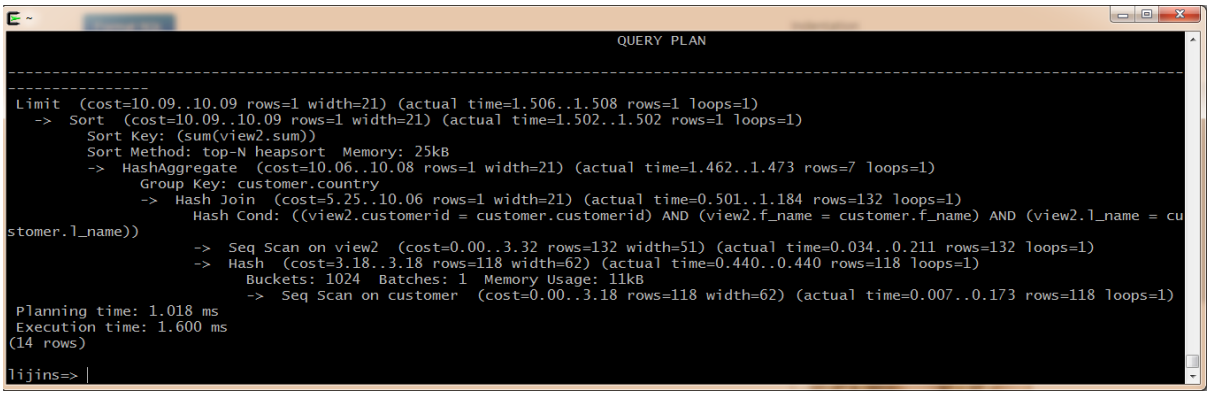
--2. The Data Mart,

```
EXPLAIN ANALYZE
SELECT country, sum(amnt)
FROM customer_dim
NATURAL JOIN sales_fact
GROUP BY country
ORDER BY sum(amnt) DESC
LIMIT 1;
```



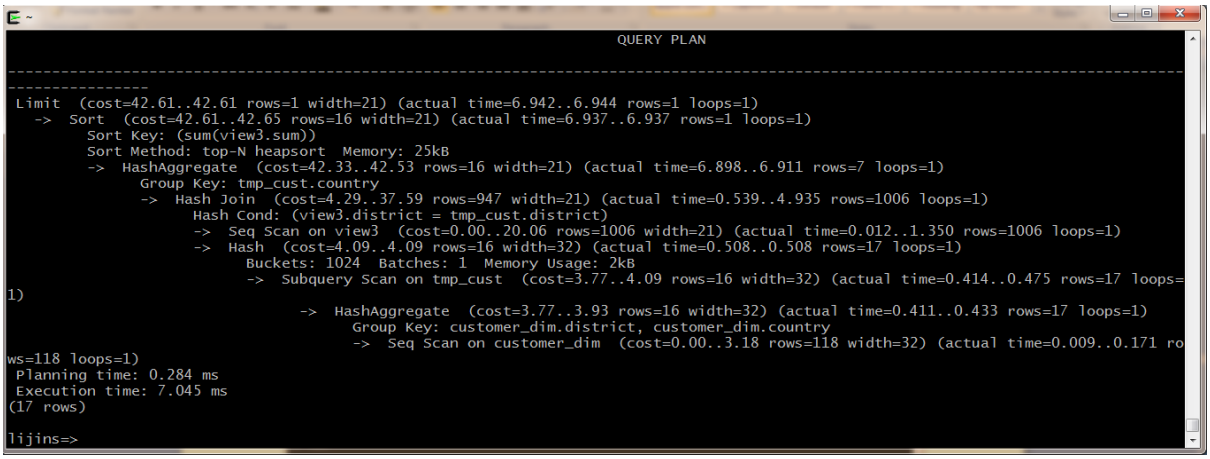
--3. The view View2

```
EXPLAIN ANALYZE
SELECT country, sum(sum)
FROM View2
NATURAL JOIN customer_dim
GROUP BY country
ORDER BY sum(sum) DESC
LIMIT 1;
```



--4. The view View3.

```
EXPLAIN ANALYZE
SELECT country, sum(sum)
FROM View3
NATURAL JOIN
 (SELECT DISTINCT district, country FROM customer_dim) AS tmp_cust
GROUP BY country
ORDER BY sum(sum) DESC
LIMIT 1;
```



Explain the findings:

1) The speed of the execution of the query "2. The Data Mata" is faster than "1. The Book Orders Database". The reason is that the sales table already contains customerid and amnt information, which reduces the join time.

2) The speed of the execution of the query "3.The view View2" is faster than "2. The Data Mata ". Because materialized views improve query performance by pre-calculating expensive join and aggregation operations.

3) The speed of the execution of the query "4.The view View3" is slower than "3. The view View2 ". We cannot use view3 directly because "district" is not a primary key, when joining to customer table, view3 needs to be modified.

Q5)-a)Display sum ammounts per customer and avg ammount per customer in a city.

```
SELECT customerid, f_name, city, sum_amnt,
    avg(sum_amnt) OVER (PARTITION BY city) AS AVG
FROM
 (SELECT customerid, f_name, city, sum(amnt) AS sum_amnt
  FROM sales_fact
  NATURAL JOIN customer_dim
  NATURAL JOIN time_dim
  WHERE (MONTH= 'April' OR MONTH= 'May') AND YEAR=2017
  GROUP BY customerid, f_name, city) AS tmp
ORDER BY city;
```

```
lijins-> ORDER BY city;
 customerid |        f_name        |      city       | sum_amnt |          avg
------------+----------------------+-----------------+----------+-----------------------
         94 | Shweta               | Auckland        |  3615.00 | 2370.0000000000000000
        113 | Tao                  | Auckland        |  2055.00 | 2370.0000000000000000
         95 | Priyanka             | Auckland        |  1440.00 | 2370.0000000000000000
        116 | Adrian               | Beijing         |   765.00 | 1157.5000000000000000
        107 | Xiaoxing             | Beijing         |  1550.00 | 1157.5000000000000000
        100 | Zoltan               | Budapest        |  2710.00 | 2710.0000000000000000
         99 | Nathan               | Christchurch    |   785.00 | 1258.3333333333333333
         98 | Valerie              | Christchurch    |   925.00 | 1258.3333333333333333
        115 | Christopher          | Christchurch    |  2065.00 | 1258.3333333333333333
        108 | Aaron                | Lower Hutt      |  1555.00 | 1555.0000000000000000
        118 | Daniel               | Masterton       |  1465.00 | 1465.0000000000000000
        101 | Benjamin             | Mumbai          |  1245.00 | 1245.0000000000000000
        112 | Bilal                | Porirua         |  1120.00 | 1297.5000000000000000
         97 | Cameron              | Porirua         |  1475.00 | 1297.5000000000000000
        102 | Leila                | Sidney          |  1165.00 | 2122.5000000000000000
        103 | Mansi                | Sidney          |  3080.00 | 2122.5000000000000000
         96 | Jovan                | Skopje          |   775.00 |  775.0000000000000000
        110 | Ronni                | Upper Hutt      |  1490.00 | 1490.0000000000000000
        109 | Neel                 | Wellington      |  1440.00 | 1155.0000000000000000
        104 | Mansour              | Wellington      |  1425.00 | 1155.0000000000000000
        114 | Harman               | Wellington      |   395.00 | 1155.0000000000000000
        111 | Kaszandra            | Wellington      |  1360.00 | 1155.0000000000000000
        105 | Jessie               | Wuhan           |  1670.00 | 1776.6666666666666667
        117 | Lei                  | Wuhan           |  1625.00 | 1776.6666666666666667
        106 | Li                   | Wuhan           |  2035.00 | 1776.6666666666666667
(25 rows)

lijins=>
```

Q5-b) Display daily sum and cumulative sum by orderdate in a city.

```
SELECT DISTINCT city, timeid, orderdate AS day, daily_sum, cumulative_sum
FROM
 (SELECT city, timeid, orderdate,
          sum(amnt) OVER w1 AS daily_sum,
          sum(amnt) OVER w2 AS cumulative_sum
  FROM sales_fact
  NATURAL JOIN time_dim
  NATURAL JOIN customer_dim
  WHERE (MONTH = 'April' OR MONTH = 'May') AND YEAR=2017
    WINDOW w1 AS (PARTITION BY city, orderdate),
             w2 AS (PARTITION BY city ORDER BY orderdate) ) AS tmp
ORDER BY city, orderdate;
```

city	timeid	day	daily_sum	cumulative_sum
Auckland	119	2017-04-23	360.00	360.00
Auckland	120	2017-04-29	2250.00	2610.00
Auckland	121	2017-04-30	2805.00	5415.00
Auckland	122	2017-05-05	1695.00	7110.00
Beijing	117	2017-04-21	1550.00	1550.00
Beijing	124	2017-05-15	765.00	2315.00
Budapest	111	2017-04-15	2710.00	2710.00
Christchurch	110	2017-04-14	925.00	925.00
Christchurch	111	2017-04-15	785.00	1710.00
Christchurch	122	2017-05-05	175.00	1885.00
Christchurch	123	2017-05-06	1810.00	3695.00
Christchurch	124	2017-05-15	80.00	3775.00
Lower Hutt	118	2017-04-22	1555.00	1555.00
Masterton	124	2017-05-15	1465.00	1465.00
Mumbai	112	2017-04-16	1245.00	1245.00
Porirua	108	2017-04-12	170.00	170.00
Porirua	109	2017-04-13	1135.00	1305.00
Porirua	110	2017-04-14	170.00	1475.00
Porirua	122	2017-05-05	1120.00	2595.00
Sidney	112	2017-04-16	850.00	850.00
Sidney	113	2017-04-17	1305.00	2155.00
Sidney	114	2017-04-18	2090.00	4245.00
Skopje	108	2017-04-12	775.00	775.00
Upper Hutt	118	2017-04-22	1490.00	1490.00
Wellington	114	2017-04-18	1425.00	1425.00
Wellington	118	2017-04-22	1440.00	2865.00
Wellington	119	2017-04-23	1360.00	4225.00
Wellington	123	2017-05-06	395.00	4620.00
Wuhan	115	2017-04-19	1735.00	1735.00
Wuhan	116	2017-04-20	1525.00	3260.00
Wuhan	117	2017-04-21	195.00	3455.00
Wuhan	118	2017-04-22	250.00	3705.00
Wuhan	124	2017-05-15	1625.00	5330.00

(33 rows)