SWEN432 Assignment 2

Question 1. A Data Mart in the PostgreSQL

1. The Customer and Book dimensions are from “Book Orders Database”:
Query: psql -d swen432_assignment2_tengzhang -f BookOrdersDatabaseDump_23.sql
Screenshots:

```
greta-pt% psql -d swen432_assignment2_tengzhang -f BookOrdersDatabaseDump_23.sql
SET
SET
SET
SET
SET
SET
SET
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
COPY 14
COPY 12
COPY 18
COPY 338
COPY 186
COPY 1516
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
```

```
swen432_assignment2_tengzhang=> \dt
              List of relations
 Schema |     Name     | Type  |   Owner
--------+--------------+-------+------------
 public | author       | table | zhangteng2
 public | book         | table | zhangteng2
 public | book_author  | table | zhangteng2
 public | cust_order   | table | zhangteng2
 public | customer     | table | zhangteng2
 public | order_detail | table | zhangteng2
(6 rows)
```

2. The Time dimension
Query and Screenshots:
CREATE TABLE time (TimeId SERIAL PRIMARY KEY, OrderDate DATE NOT NULL, DayOfWeek VARCHAR(10) NOT NULL, Month VARCHAR(10) NOT NULL,Year INT NOT NULL);

```
swen432_assignment2_tengzhang=> CREATE TABLE time (TimeId SERIAL PRIMARY KEY, OrderDate DATE NOT NULL, DayOfWeek VARCHAR(10) NOT
 NULL, Month VARCHAR(10) NOT NULL,Year INT NOT NULL);
CREATE TABLE
```

INSERT INTO Time (OrderDate, DayOfWeek, Month, Year) SELECT DISTINCT OrderDate, TO_CHAR(OrderDate, 'Day') AS DayOfWeek, TO_CHAR(OrderDate,

'Month') AS Month, EXTRACT(YEAR FROM OrderDate) AS Year FROM Cust_Order ORDER BY OrderDate ASC

```
swen432_assignment2_tengzhang=> INSERT INTO Time (OrderDate, DayOfWeek, Month, Year) SELECT DISTINCT OrderDate, TO_CHAR(OrderDat
e, 'Day') AS DayOfWeek, TO_CHAR(OrderDate, 'Month') AS Month, EXTRACT(YEAR FROM OrderDate) AS Year FROM Cust_Order ORDER BY Orde
rDate ASC;
INSERT 0 184
```

```
swen432_assignment2_tengzhang=> select * from time;
 timeid | orderdate  | dayofweek |   month   | year
--------+------------+-----------+-----------+------
      1 | 1998-03-01 | Sunday    | March     | 1998
      2 | 1998-03-21 | Saturday  | March     | 1998
      3 | 1999-01-01 | Friday    | January   | 1999
      4 | 1999-01-08 | Friday    | January   | 1999
      5 | 1999-01-09 | Saturday  | January   | 1999
      6 | 1999-01-10 | Sunday    | January   | 1999
      7 | 1999-01-11 | Monday    | January   | 1999
      8 | 1999-01-12 | Tuesday   | January   | 1999
      9 | 1999-02-08 | Monday    | February  | 1999
     10 | 1999-03-01 | Monday    | March     | 1999
     11 | 1999-03-08 | Monday    | March     | 1999
     12 | 1999-03-09 | Tuesday   | March     | 1999
     13 | 1999-03-10 | Wednesday | March     | 1999
     14 | 1999-03-11 | Thursday  | March     | 1999
     15 | 1999-03-12 | Friday    | March     | 1999
     16 | 1999-04-10 | Saturday  | April     | 1999
     17 | 1999-05-11 | Tuesday   | May       | 1999
     18 | 1999-05-12 | Wednesday | May       | 1999
     19 | 1999-07-09 | Friday    | July      | 1999
```

3. The Sales dimension

Query and Screenshots:

CREATE TABLE sales (customerid INT REFERENCES customer(customerid) CHECK (customerid > 0), timeid INT REFERENCES time(timeid) CHECK (timeid > 0), isbn INT REFERENCES book(isbn) CHECK (isbn > 0), Amnt DECIMAL(6, 2) NOT NULL CHECK (Amnt > 0), PRIMARY KEY (customerid, timeid, isbn))

```
swen432_assignment2_tengzhang=> CREATE TABLE sales (customerid INT REFERENCES customer(customerid) CHECK (customerid > 0), timei
d INT REFERENCES time(timeid) CHECK (timeid > 0), isbn INT REFERENCES book(isbn) CHECK (isbn > 0), Amnt DECIMAL(6, 2) NOT NULL C
HECK (Amnt > 0), PRIMARY KEY (customerid, timeid, isbn));
CREATE TABLE
swen432_assignment2_tengzhang=>
```

INSERT INTO Sales (CustomerId, TimeId, ISBN, Amnt) SELECT co.CustomerId, t.TimeId, b.ISBN, SUM(od.Quantity * b.Price) AS Amnt FROM Cust_Order co JOIN Order_Detail od ON co.OrderId = od.OrderId JOIN Book b ON od.ISBN = b.ISBN JOIN Time t ON t.OrderDate = co.OrderDate GROUP BY co.CustomerId, t.TimeId, b.ISBN

```
swen432_assignment2_tengzhang=> INSERT INTO Sales (CustomerId, TimeId, ISBN, Amnt) SELECT co.CustomerId, t.TimeId, b.ISBN, SUM(o
d.Quantity * b.Price) AS Amnt FROM Cust_Order co JOIN Order_Detail od ON co.OrderId = od.OrderId JOIN Book b ON od.ISBN = b.ISBN
 JOIN Time t ON t.OrderDate = co.OrderDate GROUP BY co.CustomerId, t.TimeId, b.ISBN;
INSERT 0 1469
swen432_assignment2_tengzhang=>
```

```
swen432_assignment2_tengzhang=> select * from sales;
 customerid | timeid | isbn |  amnt
------------+--------+------+--------
         78 |     52 | 5555 |  75.00
        134 |    135 | 4444 | 300.00
        112 |    122 | 4444 |  60.00
        101 |    112 | 2222 |  50.00
         79 |     61 | 4004 | 210.00
          2 |     10 | 8888 |  80.00
          1 |     14 | 1111 |  75.00
          2 |     10 | 5555 |  75.00
        122 |    126 | 4444 |  60.00
          5 |     24 | 7777 |  65.00
        130 |    135 | 6666 |  65.00
         99 |    111 | 1111 |  75.00
        102 |    113 | 2002 |  90.00
         85 |     94 | 1001 |  95.00
        138 |    139 | 4004 | 210.00
         95 |    121 | 6666 |  65.00
          3 |     23 | 2222 | 250.00
```

Question 2. Aggregate Queries

a) Conclusion: Both queries are wrong.

It can be proved with a very simple example. Assume that the current sales table stores the following records.

customerid	timeid	isbn	amnt
1	1	1111	300
1	1	2222	200
2	1	1111	300

This can be seen as the bookstore just opened and only two customers placed an order on the first day, customer #1 placed two orders for two books, and customer #2 bought one book.

Then the original problem can be simplified to: To find the average amount of money spent by all customers (2 customers) buying books on all days so far (1 day).

On this day, a total of two customers bought books, one spent 500(300 + 200) and the other spent 300. Therefore, the average amount of money spent is (500 + 300)/2 = 400.

However, the result obtained with the first query method should be:
customerid =1, avg_amnt = 250; customerid =2, avg_amnt = 300, avg(avg_amnt) = (300 + 250)/2 = 275

The result obtained with the second query method should be:
(300 + 200 + 300)/3 = 266.7

So both are wrong.

The second query is wrong because it does not consider “a customer may issue more than one order on the same date.” Therefore, the denominator is wrong. The first query error is because SUM aggregate functions should be used instead of AVG in materialized view.

b) Conclusion: The first query is wrong, the second is correct.

The purpose of the query is to find the three bestselling books overall sold in the following three cities: Auckland, Beijing, and Wellington so far.

In the first query, the materialized view displays the top three best-selling books separately in Auckland, Beijing, and Wellington, and then finds the top three best-selling books among these 9 books. This algorithm is different from the question requirements. Here is a counterexample:

City	Isbn	sum(quantity)
Auckland	1000	100
Auckland	1001	90
Auckland	1002	80
Auckland	1003	50
Beijing	1004	100
Beijing	1005	90
Beijing	1006	80
Beijing	1003	50
Wellington	1007	100
Wellington	1008	90
Wellington	1009	80
Wellington	1003	50

In the above example, the instances with green background are the results of the query for materialized view, which means that the results of query 1 will only be born from the green 9. However, actually Isbn=1003 bought 150 copies in three cities, which should be the best-selling book.

The books sold in the three cities should be sorted together, which is why the second query is correct.

Question 3. An OLAP Query

a)
SELECT CustomerID, (f_name || ' ' || l_name) AS Names, l_name AS Surname, SUM(Amnt) AS Totals FROM Sales NATURAL JOIN Customer NATURAL JOIN Time WHERE year = 2023 GROUP BY CustomerID, Names, Surname ORDER BY Totals DESC LIMIT 6

```
swen432_assignment2_tengzhang=> SELECT CustomerID, (f_name || ' ' || l_name) AS Names, l_name AS Surname,
 SUM(Amnt) AS Totals FROM Sales NATURAL JOIN Customer NATURAL JOIN Time WHERE year = 2023 GROUP BY Custom
erID, Names, Surname ORDER BY Totals DESC LIMIT 6;
 customerid |     names      |       surname        | totals
------------+----------------+----------------------+--------
        164 | Gurpreet Kaur  | Kaur                 | 515.00
        167 | Yi Sian Lim    | Lim                  | 415.00
        165 | Maegan Kennedy | Kennedy              | 285.00
        174 | Ana Ramirez    | Ramirez              | 250.00
        172 | Chintan Patel  | Patel                | 175.00
        177 | Kristen Tait   | Tait                 | 130.00
(6 rows)
```

b) Two OLAP specific operations:
Roll-Up/GROUP BY: This is a one dimensional hierarchical roll-up implemented via GROUP BY.

ORDER BY: Implementation of the rank.

Question 4. Queries Against Materialized Views

a)
1. The “Book Orders Database” (i.e., the operational database),
Query:
EXPLAIN ANALYZE SELECT CustomerID, (f_name || ' ' || l_name) AS Names, l_name AS Surname, SUM(price*quantity) AS Totals FROM Customer NATURAL JOIN cust_order NATURAL JOIN order_detail NATURAL JOIN book WHERE EXTRACT(YEAR FROM OrderDate) = 2023 GROUP BY CustomerID, Names, Surname ORDER BY Totals DESC LIMIT 6

Planning Time: 1.865 ms
Execution Time: 3.002 ms

```
swen432_assignment2_tengzhang=> SELECT CustomerID, (f_name || ' ' || l_name) AS Names, l_name AS Surname, SUM(price*quantity) AS Totals FROM Customer NATU
RAL JOIN cust_order NATURAL JOIN order_detail NATURAL JOIN book WHERE EXTRACT(YEAR FROM OrderDate) = 2023 GROUP BY CustomerID, Names, Surname ORDER BY Tot
als DESC LIMIT 6;
 customerid |     names      |       surname        | totals
------------+----------------+----------------------+--------
        164 | Gurpreet Kaur  | Kaur                 | 515.00
        167 | Yi Sian Lim    | Lim                  | 415.00
        165 | Maegan Kennedy | Kennedy              | 285.00
        174 | Ana Ramirez    | Ramirez              | 250.00
        172 | Chintan Patel  | Patel                | 175.00
        177 | Kristen Tait   | Tait                 | 130.00
(6 rows)
```

```
swen432_assignment2_tengzhang=> EXPLAIN ANALYZE SELECT CustomerID, (f_name || ' ' || l_name) AS Names, l_name AS Surname, SUM(price*quantity) AS Totals FR
OM Customer NATURAL JOIN cust_order NATURAL JOIN order_detail NATURAL JOIN book WHERE EXTRACT(YEAR FROM OrderDate) = 2023 GROUP BY CustomerID, Names, Surn
ame ORDER BY Totals DESC LIMIT 6;
                                                                     QUERY PLAN

------------------------------------------------------------------------------------------------------------------------------------------------------
-----
 Limit  (cost=38.45..38.46 rows=6 width=89) (actual time=2.594..2.638 rows=6 loops=1)
   ->  Sort  (cost=38.45..38.47 rows=9 width=89) (actual time=2.591..2.621 rows=6 loops=1)
         Sort Key: (sum((book.price * (order_detail.quantity)::numeric))) DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  GroupAggregate  (cost=37.97..38.30 rows=9 width=89) (actual time=2.202..2.469 rows=39 loops=1)
               Group Key: customer.customerid, ((((customer.f_name)::text || ' '::text) || (customer.l_name)::text))
               ->  Sort  (cost=37.97..37.99 rows=9 width=73) (actual time=2.188..2.267 rows=54 loops=1)
                     Sort Key: customer.customerid, ((((customer.f_name)::text || ' '::text) || (customer.l_name)::text))
                     Sort Method: quicksort  Memory: 29kB
                     ->  Nested Loop  (cost=11.54..37.82 rows=9 width=73) (actual time=0.749..2.065 rows=54 loops=1)
                           ->  Nested Loop  (cost=11.40..36.31 rows=9 width=52) (actual time=0.703..1.654 rows=54 loops=1)
                                 ->  Hash Join  (cost=7.09..13.90 rows=2 width=50) (actual time=0.686..0.874 rows=54 loops=1)
                                       Hash Cond: (customer.customerid = cust_order.customerid)
                                       ->  Seq Scan on customer  (cost=0.00..5.86 rows=186 width=46) (actual time=0.008..0.231 rows=186 loops=1)
                                       ->  Hash  (cost=7.07..7.07 rows=2 width=8) (actual time=0.276..0.279 rows=54 loops=1)
                                             Buckets: 1024  Batches: 1  Memory Usage: 11kB
                                             ->  Seq Scan on cust_order  (cost=0.00..7.07 rows=2 width=8) (actual time=0.074..0.160 rows=54 loops=1)
                                                   Filter: (EXTRACT(year FROM orderdate) = '2023'::numeric)
                                                   Rows Removed by Filter: 284
                                 ->  Bitmap Heap Scan on order_detail  (cost=4.31..11.16 rows=4 width=10) (actual time=0.006..0.008 rows=1 loops=54)
                                       Recheck Cond: (orderid = cust_order.orderid)
                                       Heap Blocks: exact=54
                                       ->  Bitmap Index Scan on order_detail_pkey  (cost=0.00..4.31 rows=4 width=0) (actual time=0.003..0.003 rows=1 loops
=54)
                                             Index Cond: (orderid = cust_order.orderid)
                           ->  Index Scan using book_pkey on book  (cost=0.14..0.16 rows=1 width=18) (actual time=0.003..0.003 rows=1 loops=54)
                                 Index Cond: (isbn = order_detail.isbn)
 Planning Time: 1.865 ms
 Execution Time: 3.002 ms
(28 rows)
```

2. The view View1

Query:

EXPLAIN ANALYZE SELECT CustomerID, (f_name || ' ' || l_name) AS Names, l_name AS Surname, SUM(Amnt) AS Totals FROM View1 WHERE Year = 2023 GROUP BY CustomerID, Names, Surname ORDER BY Totals DESC LIMIT 6

```
swen432_assignment2_tengzhang=> SELECT CustomerID, (f_name || ' ' || l_name) AS Names, l_name AS Surname, SUM(Amnt) AS Totals FROM View1 WHERE Year = 2023
 GROUP BY CustomerID, Names, Surname ORDER BY Totals DESC LIMIT 6;
 customerid |     names      |       surname        | totals
------------+----------------+----------------------+--------
        164 | Gurpreet Kaur  | Kaur                 | 515.00
        167 | Yi Sian Lim    | Lim                  | 415.00
        165 | Maegan Kennedy | Kennedy              | 285.00
        174 | Ana Ramirez    | Ramirez              | 250.00
        172 | Chintan Patel  | Patel                | 175.00
        177 | Kristen Tait   | Tait                 | 130.00
(6 rows)
```

```
swen432_assignment2_tengzhang=> EXPLAIN ANALYZE SELECT CustomerID, (f_name || ' ' || l_name) AS Names, l_name AS Surname, SUM(Amnt) AS Totals FROM View1 W
HERE Year = 2023 GROUP BY CustomerID, Names, Surname ORDER BY Totals DESC LIMIT 6;
                                                         QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Limit  (cost=38.34..38.36 rows=6 width=89) (actual time=0.473..0.498 rows=6 loops=1)
   ->  Sort  (cost=38.34..38.46 rows=47 width=89) (actual time=0.470..0.481 rows=6 loops=1)
         Sort Key: (sum(amnt)) DESC
         Sort Method: top-N heapsort  Memory: 26kB
         ->  HashAggregate  (cost=36.44..37.50 rows=47 width=89) (actual time=0.343..0.407 rows=39 loops=1)
               Group Key: customerid, (((f_name)::text || ' '::text) || (l_name)::text), l_name
               Batches: 1  Memory Usage: 48kB
               ->  Seq Scan on view1  (cost=0.00..35.90 rows=54 width=62) (actual time=0.012..0.222 rows=54 loops=1)
                     Filter: (year = 2023)
                     Rows Removed by Filter: 1415
 Planning Time: 0.175 ms
 Execution Time: 0.567 ms
(12 rows)
```

Planning Time: 0.175 ms

Execution Time: 0.567 ms

3. The view View2:

Query:

EXPLAIN ANALYZE SELECT CustomerID, (f_name || ' ' || l_name) AS Names, l_name AS Surname, SUM AS Totals FROM View2 WHERE Year = 2023 ORDER BY Totals DESC LIMIT 6

```
swen432_assignment2_tengzhang=> SELECT CustomerID, (f_name || ' ' || l_name) AS Names, l_name AS Surname, SUM AS Totals FROM View2 WHERE Year = 2023 ORDER
 BY Totals DESC LIMIT 6;
 customerid |     names      |       surname        | totals
------------+----------------+----------------------+--------
        164 | Gurpreet Kaur  | Kaur                 | 515.00
        167 | Yi Sian Lim    | Lim                  | 415.00
        165 | Maegan Kennedy | Kennedy              | 285.00
        174 | Ana Ramirez    | Ramirez              | 250.00
        172 | Chintan Patel  | Patel                | 175.00
        177 | Kristen Tait   | Tait                 | 130.00
(6 rows)
```

```
swen432_assignment2_tengzhang=> EXPLAIN ANALYZE SELECT CustomerID, (f_name || ' ' || l_name) AS Names, l_name AS Surname, SUM AS Totals FROM View2 WHERE Y
ear = 2023 ORDER BY Totals DESC LIMIT 6;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Limit  (cost=6.59..6.60 rows=6 width=62) (actual time=0.167..0.189 rows=6 loops=1)
   ->  Sort  (cost=6.59..6.69 rows=39 width=62) (actual time=0.164..0.173 rows=6 loops=1)
         Sort Key: sum DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  Seq Scan on view2  (cost=0.00..5.89 rows=39 width=62) (actual time=0.012..0.098 rows=39 loops=1)
               Filter: (year = 2023)
               Rows Removed by Filter: 161
 Planning Time: 0.157 ms
 Execution Time: 0.216 ms
(9 rows)

swen432_assignment2_tengzhang=>
```

Planning Time: 0.157 ms
Execution Time: 0.216 ms

b) Analyze the results:
Based on the combination of planning and execution time in a), sorted from smallest to largest: 3.view2(0.373ms) < 2.view1(0.742ms) < 1. The “Book Orders Database” (4.867ms)

According to the report analysis generated by explain analyze, the reason why view2 is faster than view1 are: reduced number of scan lines (54rows->39rows, actual time: 0.222->0.098), there is no HashAggregate calculation time, and the sorting rows and time are reduced (47rows->39rows, actual time:0.481->0.173). At the same time, the limit execution time also becomes shorter (cost:38.34->6.59, actual time: 0.498->0.189).

The reason why view1 is faster than Book Orders Database are: there is no JOIN time-consuming, there is no two-layer Nested Loop, and the number of sorting is reduced (2->1).

C) First, the original query can be optimized according to Relational Algebra. The main basis is to execute Projection as early as possible and execute Selection as early as possible.

The optimized query is as follows:
select c.customerid, c.f_name, c.l_name, sum(s.amnt) as tot_amnt from (select timeid from time where year = 2023) t JOIN (select customerid, timeid, amnt from sales) s ON s.timeid = t.timeid JOIN (select customerid, f_name, l_name from customer) c ON c.customerid = s.customerid group by c.customerid, f_name, l_name order by tot_amnt desc limit 6;

```
swen432_assignment2_tengzhang=> select c.customerid, c.f_name, c.l_name, sum(s.amnt) as tot_amnt from (select timeid from time where year = 2023) t JOIN (
select customerid, timeid, amnt from sales) s ON s.timeid = t.timeid JOIN (select customerid, f_name, l_name from customer) c ON c.customerid = s.customer
id group by c.customerid, f_name, l_name order by tot_amnt desc limit 6;
 customerid |        f_name        |        l_name        | tot_amnt
------------+----------------------+----------------------+----------
        164 | Gurpreet             | Kaur                 |   515.00
        167 | Yi Sian              | Lim                  |   415.00
        165 | Maegan               | Kennedy              |   285.00
        174 | Ana                  | Ramirez              |   250.00
        172 | Chintan              | Patel                |   175.00
        177 | Kristen              | Tait                 |   130.00
(6 rows)
```

Secondly, depending on what is implemented in the operational database, the time dimension can be treated separately to reduce the number of joins.

Step 1: Query timeid with year 2023:
Query:
select timeid from time where year = 2023;

```
swen432_assignment2_tengzhang=> select timeid from time where year = 2023;
 timeid
--------
    143
    144
    145
    146
    147
    148
    149
    150
    151
    152
    153
    154
    155
    156
    157
    158
    159
    160
    161
    162
    163
    164
    165
    166
    167
    168
    169
    170
    171
    172
    173
    174
    175
    176
    177
    178
    179
    180
    181
    182
    183
    184
(42 rows)
```

```
swen432_assignment2_tengzhang=> explain analyze select timeid from time where year = 2023;
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on "time"  (cost=0.00..4.30 rows=42 width=4) (actual time=0.022..0.074 rows=42 loops=1)
   Filter: (year = 2023)
   Rows Removed by Filter: 142
 Planning Time: 0.048 ms
 Execution Time: 0.134 ms
(5 rows)
```

It can be seen that the Planning Time is 0.048 ms and Execution Time: 0.134 ms. At the same time, we know that the corresponding timeid range for 2023 is 143～184.

Therefore, applying the timeid direct range to the Data Mart query:
select c.customerid, c.f_name, c.l_name, sum(s.amnt) as tot_amnt from (select customerid, timeid, amnt from sales Where timeid >142 AND timeid<185) s NATURAL JOIN (select customerid, f_name, l_name from customer) c group by c.customerid, f_name, l_name order by tot_amnt desc limit 6;

```
swen432_assignment2_tengzhang=> select c.customerid, c.f_name, c.l_name, sum(s.amnt) as tot_amnt from (select customerid, timeid, amnt from sales Where ti
meid >142 AND timeid<185 ) s NATURAL JOIN (select customerid, f_name, l_name from customer) c group by c.customerid, f_name, l_name order by tot_amnt desc
 limit 6;
 customerid |        f_name        |        l_name        | tot_amnt
------------+----------------------+----------------------+----------
        164 | Gurpreet             | Kaur                 |   515.00
        167 | Yi Sian              | Lim                  |   415.00
        165 | Maegan               | Kennedy              |   285.00
        174 | Ana                  | Ramirez              |   250.00
        172 | Chintan              | Patel                |   175.00
        177 | Kristen              | Tait                 |   130.00
(6 rows)
```

```
swen432_assignment2_tengzhang=> explain analyze select c.customerid, c.f_name, c.l_name, sum(s.amnt) as tot_amnt from (select customerid, timeid, amnt fro
m sales Where timeid >142 AND timeid<185 ) s NATURAL JOIN (select customerid, f_name, l_name from customer) c group by c.customerid, f_name, l_name order
by tot_amnt desc limit 6;
                                                                                                                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=42.20..42.21 rows=6 width=78) (actual time=1.046..1.078 rows=6 loops=1)
   ->  Sort  (cost=42.20..42.33 rows=52 width=78) (actual time=1.043..1.061 rows=6 loops=1)
         Sort Key: (sum(sales.amnt)) DESC
         Sort Method: top-N heapsort  Memory: 26kB
         ->  HashAggregate  (cost=40.62..41.27 rows=52 width=78) (actual time=0.917..0.987 rows=39 loops=1)
               Group Key: customer.customerid
               Batches: 1  Memory Usage: 48kB
               ->  Hash Join  (cost=8.19..40.36 rows=52 width=51) (actual time=0.514..0.824 rows=54 loops=1)
                     Hash Cond: (sales.customerid = customer.customerid)
                     ->  Seq Scan on sales  (cost=0.00..32.03 rows=52 width=9) (actual time=0.010..0.175 rows=54 loops=1)
                           Filter: ((timeid > 142) AND (timeid < 185))
                           Rows Removed by Filter: 1415
                     ->  Hash  (cost=5.86..5.86 rows=186 width=46) (actual time=0.495..0.499 rows=186 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 23kB
                           ->  Seq Scan on customer  (cost=0.00..5.86 rows=186 width=46) (actual time=0.007..0.240 rows=186 loops=1)
 Planning Time: 0.230 ms
 Execution Time: 1.123 ms
(17 rows)
```

From the above figure, the query results are the same, but the query time is Planning Time is 0.230ms and Execution Time is 1.123ms.

In total, Planning Time = 0.23+0.048=0.278ms, Execution Time=1.123+0.134-1.257ms. This result is three times faster than the query in the question.

In addition, indexes can be added to the customer and sales tables to further speed up the process, but I am not sure whether this counts as changing the structure of Data Mart, so it has not been implemented.

D)
1. The Data Mart:
EXPLAIN ANALYZE SELECT c.Country, SUM(s.Amnt) AS Total FROM Customer c NATURAL JOIN Sales s GROUP BY c.Country ORDER BY Total DESC LIMIT 1;

```
swen432_assignment2_tengzhang=>  SELECT c.Country, SUM(s.Amnt) AS Total FROM Customer c NATURAL JOIN Sales s GROUP BY c.Country ORDER BY Total DESC LIMIT
1;
     country     |   total
-----------------+-----------
 New Zealand     | 158475.00
(1 row)
```

```
swen432_assignment2_tengzhang=> EXPLAIN ANALYZE SELECT c.Country, SUM(s.Amnt) AS Total FROM Customer c NATURAL JOIN Sales s GROUP BY c.Country ORDER BY To
tal DESC LIMIT 1;
                                                                                                                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=44.31..44.31 rows=1 width=48) (actual time=8.164..8.179 rows=1 loops=1)
   ->  Sort  (cost=44.31..44.33 rows=8 width=48) (actual time=8.162..8.173 rows=1 loops=1)
         Sort Key: (sum(s.amnt)) DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  HashAggregate  (cost=44.17..44.27 rows=8 width=48) (actual time=8.133..8.154 rows=8 loops=1)
               Group Key: c.country
               Batches: 1  Memory Usage: 24kB
               ->  Hash Join  (cost=8.19..36.82 rows=1469 width=21) (actual time=0.517..5.691 rows=1469 loops=1)
                     Hash Cond: (s.customerid = c.customerid)
                     ->  Seq Scan on sales s  (cost=0.00..24.69 rows=1469 width=9) (actual time=0.007..1.546 rows=1469 loops=1)
                     ->  Hash  (cost=5.86..5.86 rows=186 width=20) (actual time=0.501..0.504 rows=186 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 18kB
                           ->  Seq Scan on customer c  (cost=0.00..5.86 rows=186 width=20) (actual time=0.008..0.248 rows=186 loops=1)
 Planning Time: 0.181 ms
 Execution Time: 8.216 ms
(15 rows)
```

Planning Time: 0.181 ms
Execution Time: 8.216 ms

2. The view View2:

EXPLAIN ANALYZE SELECT c.Country, SUM(v.SUM) AS Total FROM View2 v NATURAL JOIN Customer c GROUP BY c.Country ORDER BY Total DESC LIMIT 1;

```
swen432_assignment2_tengzhang=>  SELECT c.Country, SUM(v.SUM) AS Total FROM View2 v NATURAL JOIN Customer c GROUP BY c.Country ORDER BY Total DESC LIMIT 1
;
     country     |   total
-----------------+-----------
 New Zealand     | 158475.00
(1 row)
```

```
swen432_assignment2_tengzhang=> EXPLAIN ANALYZE SELECT c.Country, SUM(v.SUM) AS Total FROM View2 v NATURAL JOIN Customer c GROUP BY c.Country ORDER BY Tot
al DESC LIMIT 1;
                                                              QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=15.73..15.74 rows=1 width=48) (actual time=2.176..2.192 rows=1 loops=1)
   ->  Sort  (cost=15.73..15.74 rows=1 width=48) (actual time=2.173..2.186 rows=1 loops=1)
         Sort Key: (sum(v.sum)) DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  GroupAggregate  (cost=15.70..15.72 rows=1 width=48) (actual time=1.720..2.168 rows=8 loops=1)
               Group Key: c.country
               ->  Sort  (cost=15.70..15.71 rows=1 width=21) (actual time=1.662..1.871 rows=200 loops=1)
                     Sort Key: c.country
                     Sort Method: quicksort  Memory: 40kB
                     ->  Hash Join  (cost=9.12..15.69 rows=1 width=21) (actual time=0.556..1.389 rows=200 loops=1)
                           Hash Cond: ((v.customerid = c.customerid) AND (v.f_name = c.f_name) AND (v.l_name = c.l_name))
                           ->  Seq Scan on view2 v  (cost=0.00..5.00 rows=200 width=51) (actual time=0.010..0.242 rows=200 loops=1)
                           ->  Hash  (cost=5.86..5.86 rows=186 width=62) (actual time=0.535..0.538 rows=186 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 26kB
                                 ->  Seq Scan on customer c  (cost=0.00..5.86 rows=186 width=62) (actual time=0.009..0.256 rows=186 loops=1)
 Planning Time: 0.282 ms
 Execution Time: 2.227 ms
(17 rows)
```

Planning Time: 0.282 ms

Execution Time: 2.227 ms

3. The view View3:

EXPLAIN ANALYZE SELECT c.country, SUM(v.SUM) AS total FROM View3 v JOIN (SELECT DISTINCT country, district FROM Customer) c ON v.district = c.district GROUP BY c.country ORDER BY total DESC LIMIT 1

```
swen432_assignment2_tengzhang=> SELECT c.country, SUM(v.SUM) AS total FROM View3 v JOIN (SELECT DISTINCT country, district FR
OM Customer) c ON v.district = c.district GROUP BY c.country ORDER BY total DESC LIMIT 1;
     country     |   total
-----------------+-----------
 New Zealand     | 158475.00
(1 row)
```

```
swen432_assignment2_tengzhang=> EXPLAIN ANALYZE SELECT c.country, SUM(v.SUM) AS total FROM View3 v JOIN (SELECT DISTINCT coun
try, district FROM Customer) c ON v.district = c.district GROUP BY c.country ORDER BY total DESC LIMIT 1;
                                                                 QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------
--------------
 Limit  (cost=62.50..62.50 rows=1 width=48) (actual time=8.026..8.044 rows=1 loops=1)
   ->  Sort  (cost=62.50..62.55 rows=19 width=48) (actual time=8.023..8.038 rows=1 loops=1)
         Sort Key: (sum(v.sum)) DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  HashAggregate  (cost=62.17..62.41 rows=19 width=48) (actual time=7.992..8.017 rows=8 loops=1)
               Group Key: customer.country
               Batches: 1  Memory Usage: 24kB
               ->  Hash Join  (cost=7.41..55.25 rows=1383 width=21) (actual time=0.588..5.861 rows=1383 loops=1)
                     Hash Cond: (v.district = customer.district)
                     ->  Seq Scan on view3 v  (cost=0.00..28.83 rows=1383 width=26) (actual time=0.009..1.473 rows=1383 loops
=1)
                     ->  Hash  (cost=7.17..7.17 rows=19 width=37) (actual time=0.571..0.577 rows=19 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 10kB
                           ->  HashAggregate  (cost=6.79..6.98 rows=19 width=37) (actual time=0.518..0.545 rows=19 loops=1)
                                 Group Key: customer.country, customer.district
                                 Batches: 1  Memory Usage: 24kB
                                 ->  Seq Scan on customer  (cost=0.00..5.86 rows=186 width=37) (actual time=0.008..0.208 rows
=186 loops=1)
 Planning Time: 0.142 ms
 Execution Time: 8.096 ms
(18 rows)
```

Planning Time: 0.142 ms

Execution Time: 8.096 ms

e) Analyze the results:
Based on the ascending value of the response time, the above query is sorted as: View2(2.509ms) < View3(8.238ms) < The Data Mart(8.397ms).

The reasons why View3 is slower than View2 are: view3 performs an extra step of HashAggregate (cost:6.98, actual time:0.545); Scan on View3 has more rows than view2(rows:1383->200, actual time: 1.473->0.242); View3 takes longer to execute Hash Join than View2 (cost:55.25->15.69, actual time:5.861->1.389); View3 has a longer sort cost time than view2 (cost:62.55->15.74, actual time:8.038->2.186); View3 has a longer Limit cost time than view2(cost:62.5->15.74, actual time:8.044->2.192).

The reasons why The Data Mart is slower than View3 are: Data Mart takes longer time to scan on customer than View3(actual time:0.248->0.208); Data Mart take longer to scan for sales than view 3 (actual time: 1.546->1.473); Data Mart executes HashAggregate longer than view3(actual time: 8.154->8.017); Data Mart takes longer to execute Sort than view3(actual time:8.173->8.038); Data Mart executes Limit longer than view3(actual time: 8.179->8.044).

Question 5. Queries with WINDOW Function

First create a MATERIALIZED VIEW and the columns required for query: city, month, sum (Amnt). At the same time, for the convenience of comparison, sort the months in city partition by AVG (timeid) (as the timeid is sorted by date, so the same year AVG (January timeid) is always smaller than AVG (February timeid))

CREATE MATERIALIZED VIEW View4 AS SELECT City, Month, SUM(Amnt) AS Sum_Amunt, AVG(timeid) AS Avg_TimeId FROM Sales NATURAL JOIN Customer NATURAL JOIN Time Where Year = 2023 GROUP BY City, Month Order by City, Avg_TimeId

```
swen432_assignment2_tengzhang=> CREATE MATERIALIZED VIEW View4 AS SELECT City, Month, SUM(Amnt) AS Sum_Amunt, AVG(timeid) AS
 Avg_TimeId FROM Sales NATURAL JOIN Customer NATURAL JOIN Time Where Year = 2023 GROUP BY City, Month Order by City, Avg_Tim
eId;
SELECT 17
swen432_assignment2_tengzhang=> select * from view4;
      city       |   month   | sum_amunt |      avg_timeid
-----------------+-----------+-----------+----------------------
 Auckland        | January   |    450.00 | 146.0000000000000000
 Auckland        | February  |    155.00 | 153.0000000000000000
 Auckland        | March     |    285.00 | 160.5000000000000000
 Auckland        | April     |    125.00 | 167.5000000000000000
 Auckland        | May       |    250.00 | 175.2500000000000000
 Auckland        | June      |     65.00 | 183.0000000000000000
 Beijing         | January   |    155.00 | 148.5000000000000000
 Beijing         | February  |    255.00 | 156.0000000000000000
 Beijing         | March     |    100.00 | 164.0000000000000000
 Beijing         | April     |    150.00 | 169.5000000000000000
 Beijing         | May       |    360.00 | 175.2500000000000000
 Wellington      | January   |    165.00 | 147.0000000000000000
 Wellington      | February  |    350.00 | 152.0000000000000000
 Wellington      | March     |    300.00 | 160.2500000000000000
 Wellington      | April     |    395.00 | 168.4000000000000000
 Wellington      | May       |    310.00 | 176.7500000000000000
 Wellington      | June      |    395.00 | 181.6000000000000000
(17 rows)
```

The second step is to use the yearly average amount as the window function. I initially used AVG (sum_amun) as yearly average amount of money, but since each city lacks some months sales data, it can be regarded as the sales volume of the current month is 0, so SUM(sum_amunt) /12 is used here to calculate yearly average amount of money. In addition, AVG (sum_amun) calculates the average of the months with sales.

SELECT City, Month, sum_amunt AS Monthly_Amount,
ROUND(SUM(sum_amunt) OVER w/12, 2) AS Yearly_Average_Amount FROM View4 WINDOW w AS (PARTITION BY City)

```
swen432_assignment2_tengzhang=> SELECT City, Month, sum_amunt AS Monthly_Amount, ROUND( SUM(sum_amunt) OVE
R w/12, 2) AS Yearly_Average_Amount FROM View4 WINDOW w AS (PARTITION BY City);
      city       |   month   | monthly_amount | yearly_average_amount
-----------------+-----------+----------------+-----------------------
 Auckland        | January   |         450.00 |                110.83
 Auckland        | February  |         155.00 |                110.83
 Auckland        | March     |         285.00 |                110.83
 Auckland        | April     |         125.00 |                110.83
 Auckland        | May       |         250.00 |                110.83
 Auckland        | June      |          65.00 |                110.83
 Beijing         | January   |         155.00 |                 85.00
 Beijing         | February  |         255.00 |                 85.00
 Beijing         | March     |         100.00 |                 85.00
 Beijing         | April     |         150.00 |                 85.00
 Beijing         | May       |         360.00 |                 85.00
 Wellington      | January   |         165.00 |                159.58
 Wellington      | February  |         350.00 |                159.58
 Wellington      | March     |         300.00 |                159.58
 Wellington      | April     |         395.00 |                159.58
 Wellington      | May       |         310.00 |                159.58
 Wellington      | June      |         395.00 |                159.58
(17 rows)
```