Names: Twakira Agape Gift

ID: 27320

PL/SQL Window Functions Assignment Report

1. Introduction

. Using SQL window functions, businesses can calculate rankings, running totals, moving averages, and customer segments while still keeping row-level details. This helps not only in summarizing data but also in tracking trends and diagnosing problems. In this project, the case study is **Fresh Direct**, a company that manages small micro-hubs for fresh produce distribution across Kigali and nearby regions. The aim is to use window functions to derive actionable insights from their sales data.

2. Problem Definition

Fresh Direct faces challenges such as identifying top-selling products, tracking sales trends, understanding growth rates, segmenting customers, and forecasting demand. This project applies PL/SQL window functions to provide structured solutions to these analytical problems, turning raw data into a strategic asset.

3. Success Criteria

The project was successful it answers these questions:

1. Who are the **top-5 products** per hub and quarter? ***(ROW_NUMBER, RANK, DENSE_RANK, PERCENT_RANK)***
2. What are the **running monthly sales totals**? ***(SUM OVER)***
3. How do sales change **month-to-month**? ***(LAG)***

4. How can customers be divided into **quartiles** based on spending? ***(NTILE, CUME_DIST)***
5. What are the **3-month moving averages** for product sales? ***(AVG OVER)***

4. Database Schema

Entities: Regions, Hubs, Farmers, Customers, Products, Orders, Order_Items.

ER Diagram:

an ERD for FreshDirect, which is a company that manages micro-hubs for fresh produce distribution. The database must include the following entities: regions, hubs, farmers, customers, products, orders, and order_items.

Based on the provided schema and relationships, here is a detailed ERD:

Entities and Attributes:

1. **Regions**
    - region_id (Primary Key)
    - region_name
2. **Hubs**
    - hub_id (Primary Key)
    - hub_name
    - region_id (Foreign Key to Regions)
3. **Farmers**
    - farmer_id (Primary Key)
    - farmer_name

   - region_id (Foreign Key to Regions)

4. **Customers**
   - customer_id (Primary Key)
   - full_name
   - region_id (Foreign Key to Regions)
   - join_date
5. **Products**
   - product_id (Primary Key)
   - product_name
   - category
   - unit_price
6. **Orders**
   - order_id (Primary Key)
   - customer_id (Foreign Key to Customers)
   - hub_id (Foreign Key to Hubs)
   - order_date
   - status
7. **Order_Items**
   - item_id (Primary Key)
   - order_id (Foreign Key to Orders)

- product_id (Foreign Key to Products)
- farmer_id (Foreign Key to Farmers)
- quantity
- unit_price
- amount

Relationships:

- A Region can have multiple Hubs (1:M).
- A Region can have multiple Farmers (1:M).
- A Region can have multiple Customers (1:M).
- A Hub can have multiple Orders (1:M).
- A Customer can have multiple Orders (1:M).
- An Order can have multiple Order_Items (1:M).
- A Product can be in multiple Order_Items (1:M).
- A Farmer can be associated with multiple Order_Items (1:M).

However, note that in the Order_Items table, we have a direct link to the farmer who supplied the product for that order item.

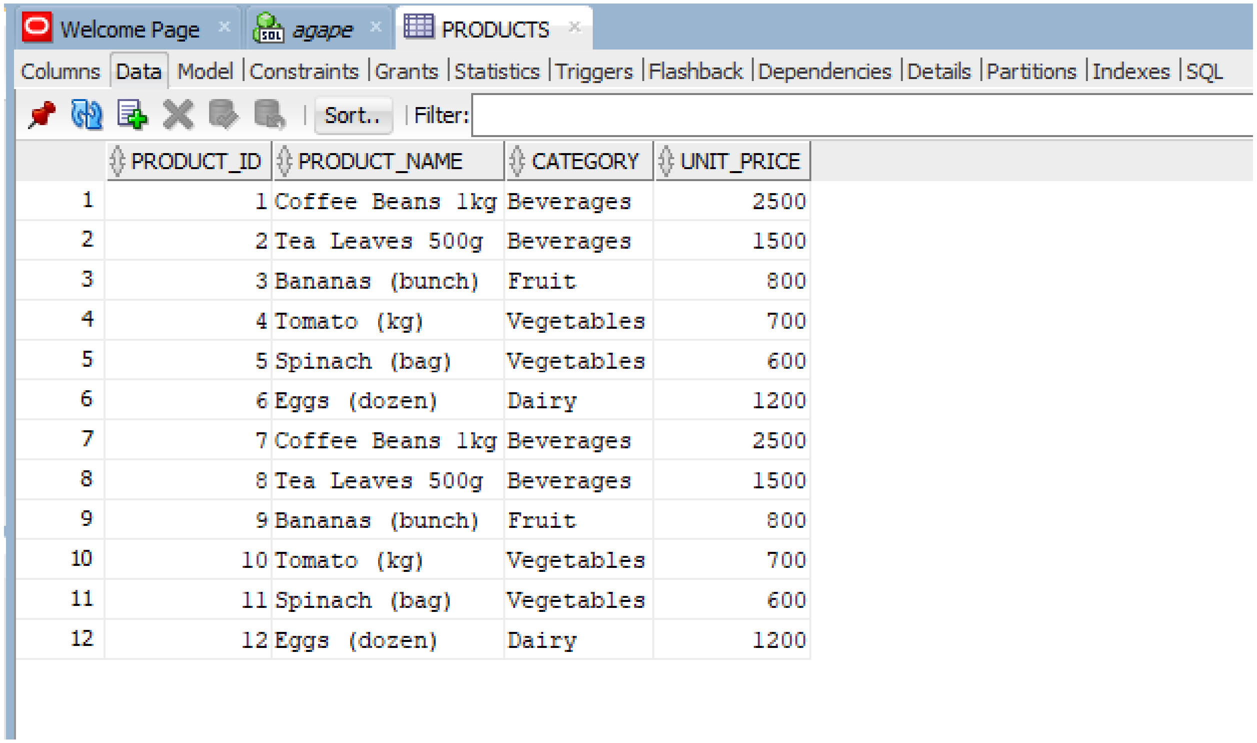
This ERD is accurate and unique to FreshDirect's business model, which connects farmers to customers through hubs, with orders being processed at hubs and each order item linked to a specific farmer.

5. Queries and Results

5.1 Top-5 Products per Hub & Quarter

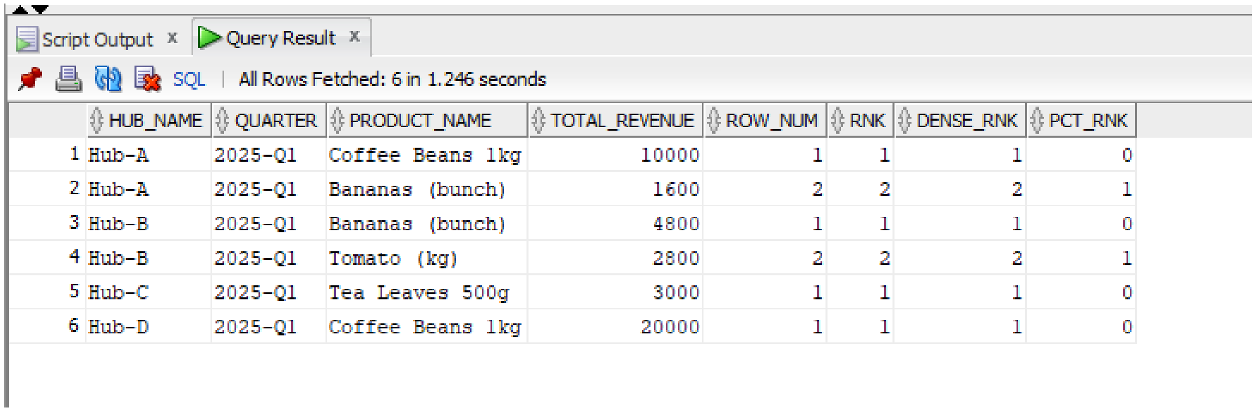
Functions used: ROW_NUMBER, RANK, DENSE_RANK, PERCENT_RANK

PRODUCTS Table:



	PRODUCT_ID	PRODUCT_NAME	CATEGORY	UNIT_PRICE
1	1	Coffee Beans 1kg	Beverages	2500
2	2	Tea Leaves 500g	Beverages	1500
3	3	Bananas (bunch)	Fruit	800
4	4	Tomato (kg)	Vegetables	700
5	5	Spinach (bag)	Vegetables	600
6	6	Eggs (dozen)	Dairy	1200
7	7	Coffee Beans 1kg	Beverages	2500
8	8	Tea Leaves 500g	Beverages	1500
9	9	Bananas (bunch)	Fruit	800
10	10	Tomato (kg)	Vegetables	700
11	11	Spinach (bag)	Vegetables	600
12	12	Eggs (dozen)	Dairy	1200

- **Screenshot:**



All Rows Fetched: 6 in 1.246 seconds

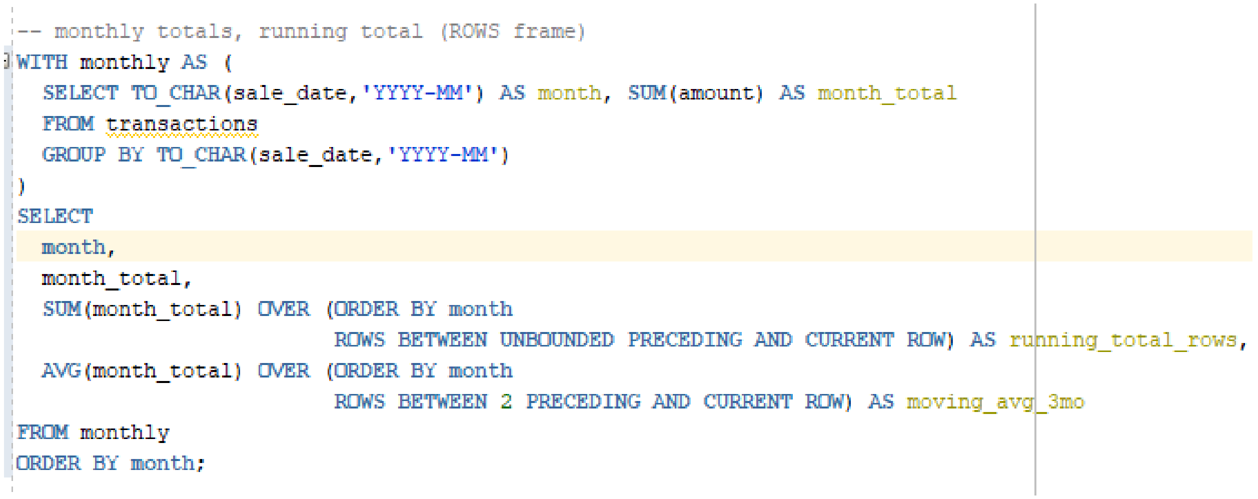
	HUB_NAME	QUARTER	PRODUCT_NAME	TOTAL_REVENUE	ROW_NUM	RNK	DENSE_RNK	PCT_RNK
1	Hub-A	2025-Q1	Coffee Beans 1kg	10000	1	1	1	0
2	Hub-A	2025-Q1	Bananas (bunch)	1600	2	2	2	1
3	Hub-B	2025-Q1	Bananas (bunch)	4800	1	1	1	0
4	Hub-B	2025-Q1	Tomato (kg)	2800	2	2	2	1
5	Hub-C	2025-Q1	Tea Leaves 500g	3000	1	1	1	0
6	Hub-D	2025-Q1	Coffee Beans 1kg	20000	1	1	1	0

- **Explanation:** This query ranks products by revenue within each hub and quarter. The output shows the best-performing products, such as "Coffee Beans 1kg" in Hub-D, which management can use to make decisions about promotions and inventory prioritization. The different ranking functions handle ties in revenue differently, providing flexibility in analysis.

5.2 Running Monthly Sales Totals

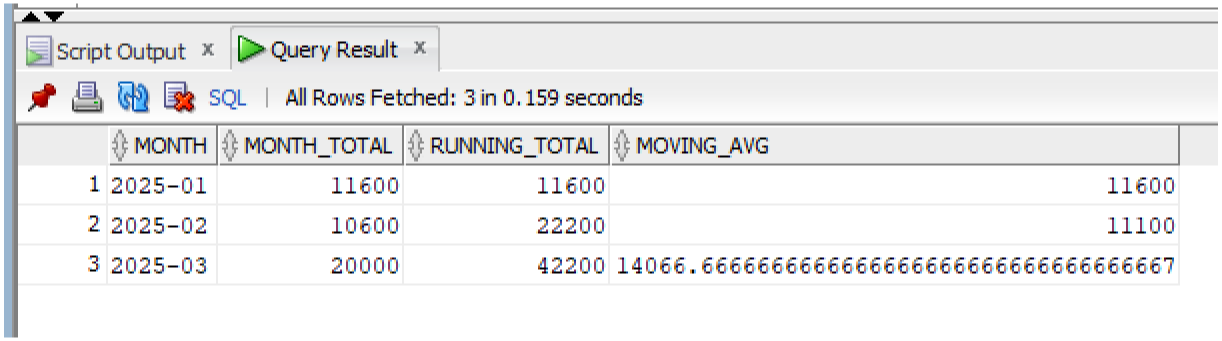
Functions used: SUM OVER

- **SQL Code:**



```sql
-- monthly totals, running total (ROWS frame)
WITH monthly AS (
  SELECT TO_CHAR(sale_date,'YYYY-MM') AS month, SUM(amount) AS month_total
  FROM transactions
  GROUP BY TO_CHAR(sale_date,'YYYY-MM')
)
SELECT
  month,
  month_total,
  SUM(month_total) OVER (ORDER BY month
                         ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS running_total_rows,
  AVG(month_total) OVER (ORDER BY month
                         ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS moving_avg_3mo
FROM monthly
ORDER BY month;
```

- **Screenshot:**



All Rows Fetched: 3 in 0.159 seconds

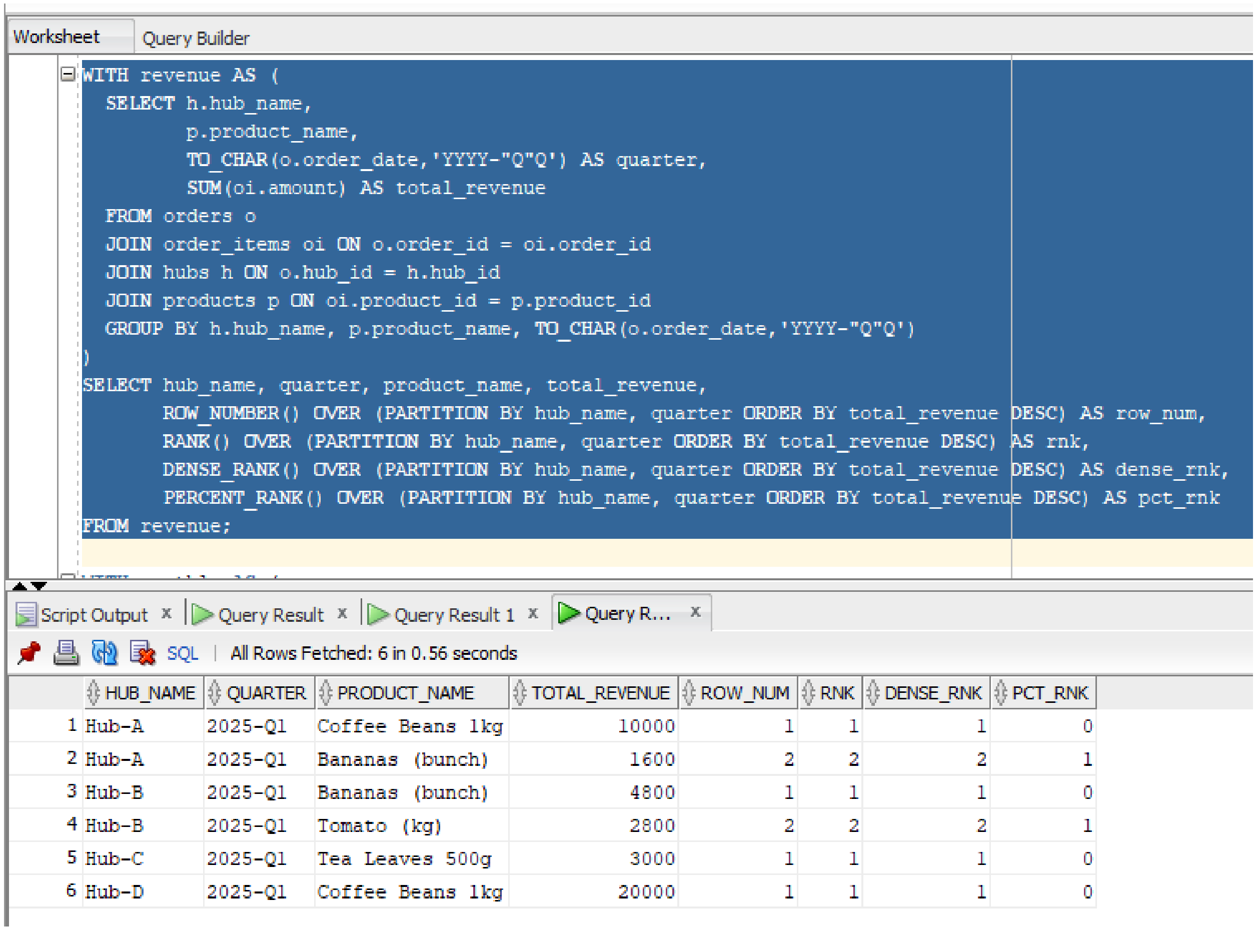
	MONTH	MONTH_TOTAL	RUNNING_TOTAL	MOVING_AVG
1	2025-01	11600	11600	11600
2	2025-02	10600	22200	11100
3	2025-03	20000	42200	14066.66666666666666666666666666666667

- **Explanation:** This query calculates the running total of sales, showing the cumulative revenue growth over time. It helps management see the overall business trajectory. The result clearly shows that sales are accumulating positively, reaching a total of 42,200 by the end of March 2025.

5.3 Month-over-Month Sales Growth

Functions used: LAG

Screenshot:

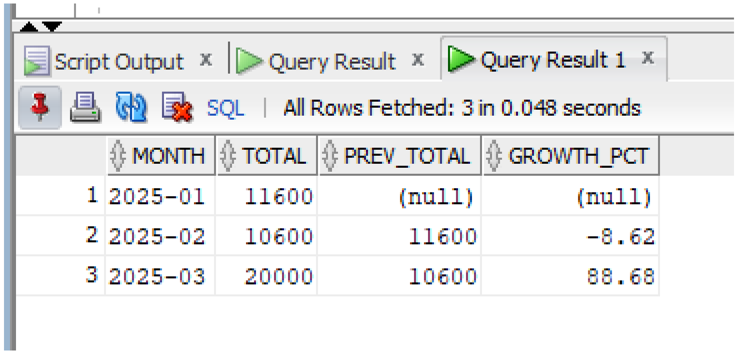


```
WITH revenue AS (
  SELECT h.hub_name,
         p.product_name,
         TO_CHAR(o.order_date,'YYYY-"Q"Q') AS quarter,
         SUM(oi.amount) AS total_revenue
  FROM orders o
  JOIN order_items oi ON o.order_id = oi.order_id
  JOIN hubs h ON o.hub_id = h.hub_id
  JOIN products p ON oi.product_id = p.product_id
  GROUP BY h.hub_name, p.product_name, TO_CHAR(o.order_date,'YYYY-"Q"Q')
)
SELECT hub_name, quarter, product_name, total_revenue,
       ROW_NUMBER() OVER (PARTITION BY hub_name, quarter ORDER BY total_revenue DESC) AS row_num,
       RANK() OVER (PARTITION BY hub_name, quarter ORDER BY total_revenue DESC) AS rnk,
       DENSE_RANK() OVER (PARTITION BY hub_name, quarter ORDER BY total_revenue DESC) AS dense_rnk,
       PERCENT_RANK() OVER (PARTITION BY hub_name, quarter ORDER BY total_revenue DESC) AS pct_rnk
FROM revenue;
```

All Rows Fetched: 6 in 0.56 seconds

	HUB_NAME	QUARTER	PRODUCT_NAME	TOTAL_REVENUE	ROW_NUM	RNK	DENSE_RNK	PCT_RNK
1	Hub-A	2025-Q1	Coffee Beans 1kg	10000	1	1	1	0
2	Hub-A	2025-Q1	Bananas (bunch)	1600	2	2	2	1
3	Hub-B	2025-Q1	Bananas (bunch)	4800	1	1	1	0
4	Hub-B	2025-Q1	Tomato (kg)	2800	2	2	2	1
5	Hub-C	2025-Q1	Tea Leaves 500g	3000	1	1	1	0
6	Hub-D	2025-Q1	Coffee Beans 1kg	20000	1	1	1	0

- **Screenshot:**



All Rows Fetched: 3 in 0.048 seconds

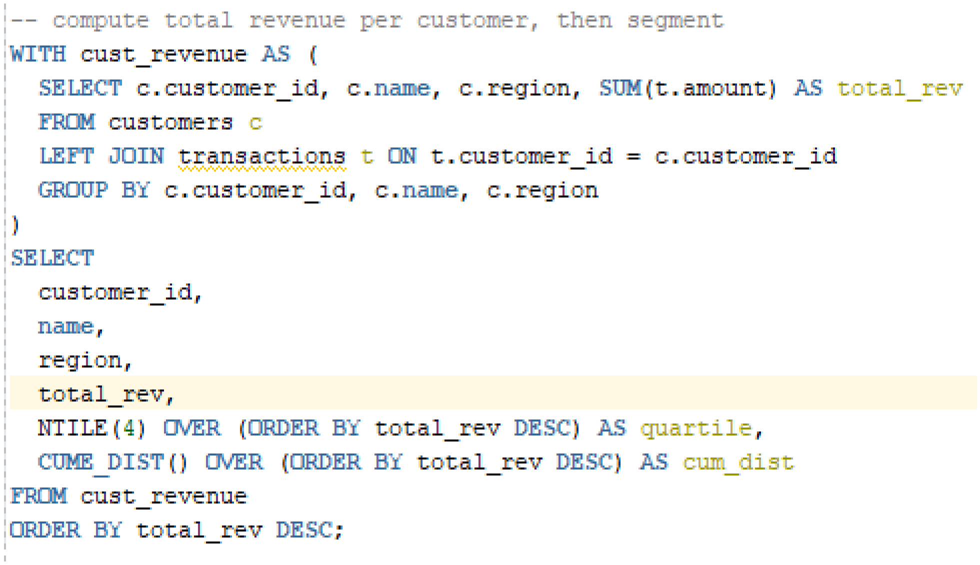
```
  MONTH    TOTAL  PREV_TOTAL  GROWTH_PCT
1 2025-01  11600      (null)      (null)
2 2025-02  10600       11600       -8.62
3 2025-03  20000       10600       88.68
```

- **Explanation:** This query compares each month's sales to the previous month, calculating a growth percentage. The output reveals important trends, such as a significant sales jump of 88.68% in March. This insight is critical for diagnosing the reasons behind successful months and replicating those conditions.

5.4 Customer Quartiles by Spending

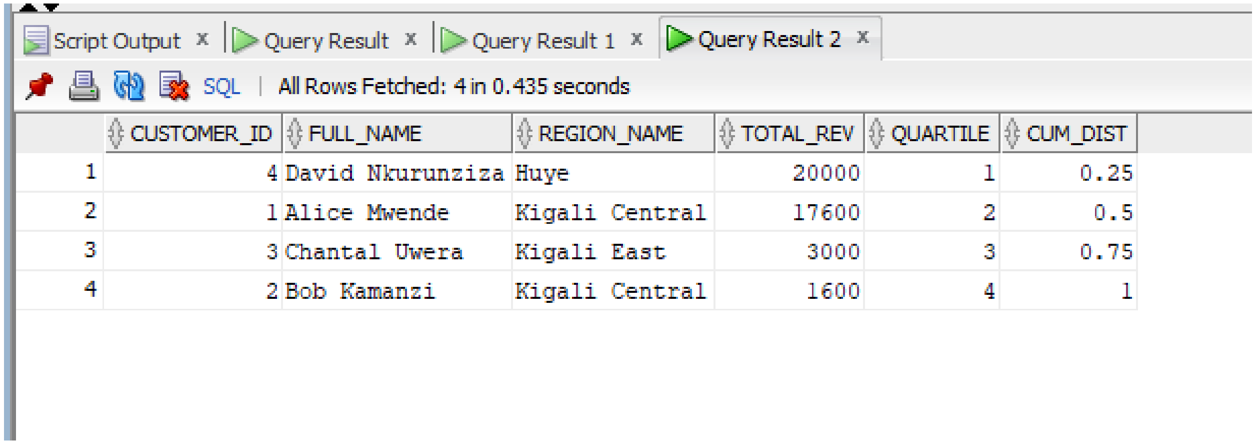
Functions used: NTILE, CUME_DIST

- **SQL Code:**



```sql
-- compute total revenue per customer, then segment
WITH cust_revenue AS (
  SELECT c.customer_id, c.name, c.region, SUM(t.amount) AS total_rev
  FROM customers c
  LEFT JOIN transactions t ON t.customer_id = c.customer_id
  GROUP BY c.customer_id, c.name, c.region
)
SELECT
  customer_id,
  name,
  region,
  total_rev,
  NTILE(4) OVER (ORDER BY total_rev DESC) AS quartile,
  CUME_DIST() OVER (ORDER BY total_rev DESC) AS cum_dist
FROM cust_revenue
ORDER BY total_rev DESC;
```

- **Screenshot:**



All Rows Fetched: 4 in 0.435 seconds

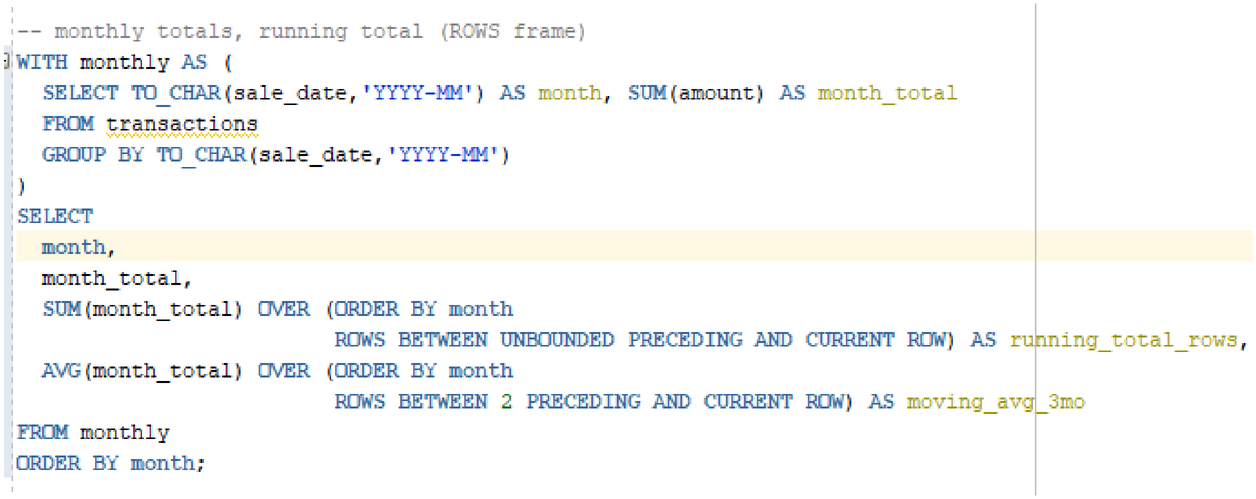
	CUSTOMER_ID	FULL_NAME	REGION_NAME	TOTAL_REV	QUARTILE	CUM_DIST
1	4	David Nkurunziza	Huye	20000	1	0.25
2	1	Alice Mwende	Kigali Central	17600	2	0.5
3	3	Chantal Uwera	Kigali East	3000	3	0.75
4	2	Bob Kamanzi	Kigali Central	1600	4	1

- **Explanation:** This query divides customers into four equal groups (quartiles) based on their total spending. Customers in the first quartile (like David) are the most valuable and should be targeted for premium loyalty programs. The `CUME_DIST` function further shows the relative standing of each customer within the spending distribution.

5.5 3-Month Moving Average of Product Sales

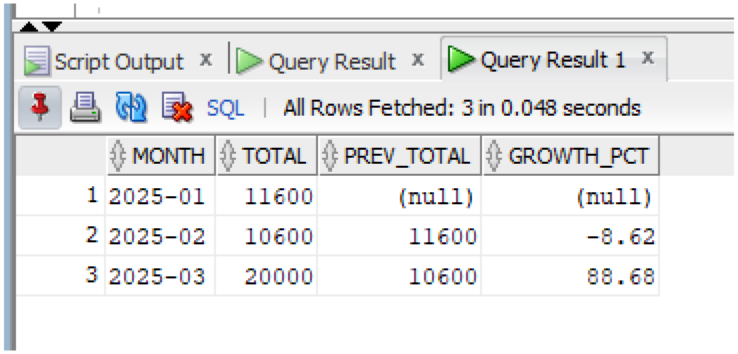
Functions used: AVG OVER

  - **SQL Code:**



```
-- monthly totals, running total (ROWS frame)
WITH monthly AS (
  SELECT TO_CHAR(sale_date,'YYYY-MM') AS month, SUM(amount) AS month_total
  FROM transactions
  GROUP BY TO_CHAR(sale_date,'YYYY-MM')
)
SELECT
  month,
  month_total,
  SUM(month_total) OVER (ORDER BY month
                         ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS running_total_rows,
  AVG(month_total) OVER (ORDER BY month
                         ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS moving_avg_3mo
FROM monthly
ORDER BY month;
```

- **Screenshot:**

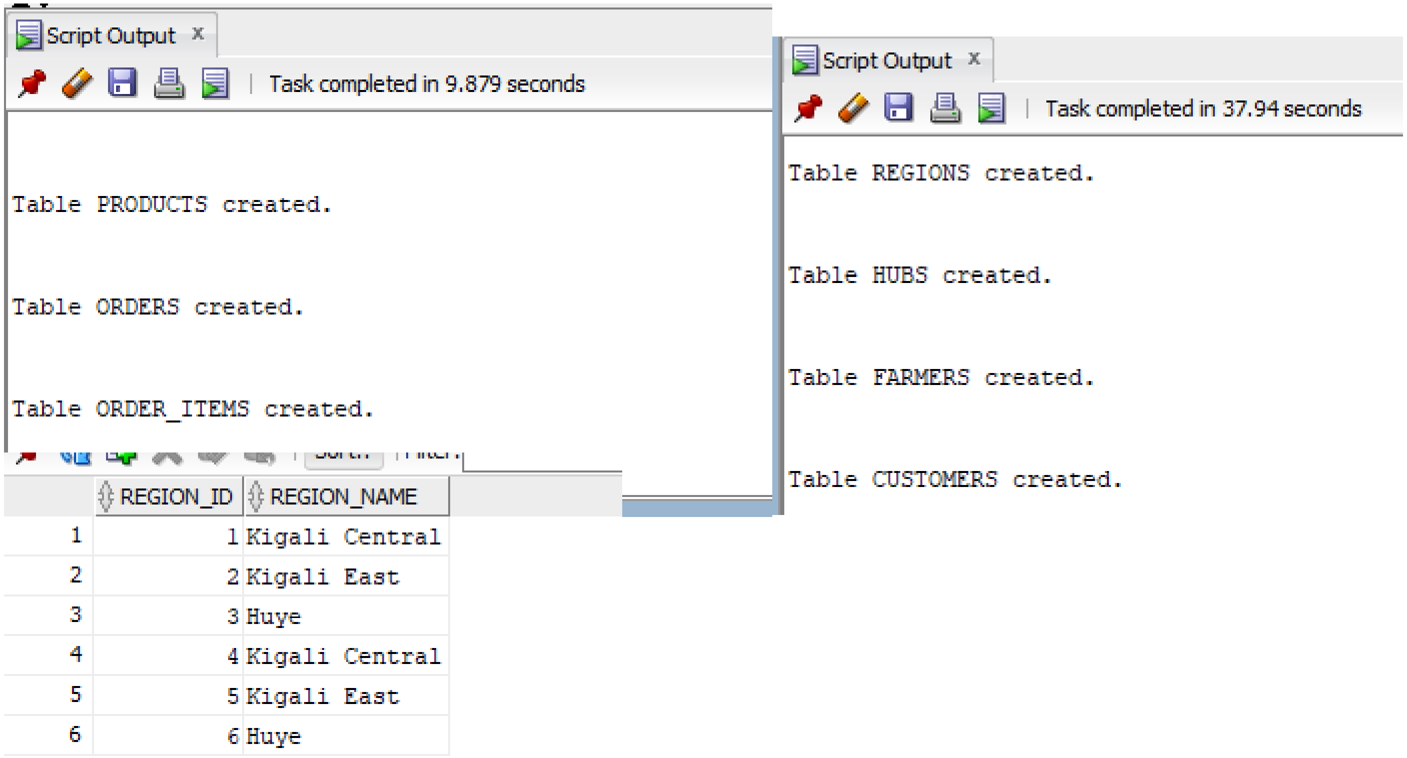


All Rows Fetched: 3 in 0.048 seconds

	MONTH	TOTAL	PREV_TOTAL	GROWTH_PCT
1	2025-01	11600	(null)	(null)
2	2025-02	10600	11600	-8.62
3	2025-03	20000	10600	88.68

- **Explanation:** The moving average smooths out short-term fluctuations in sales data, making it easier to identify long-term trends for each product. This helps in making more accurate inventory and purchasing decisions by focusing on the underlying trend rather than monthly spikes or dips.

All Other srceenshots to show more about the work



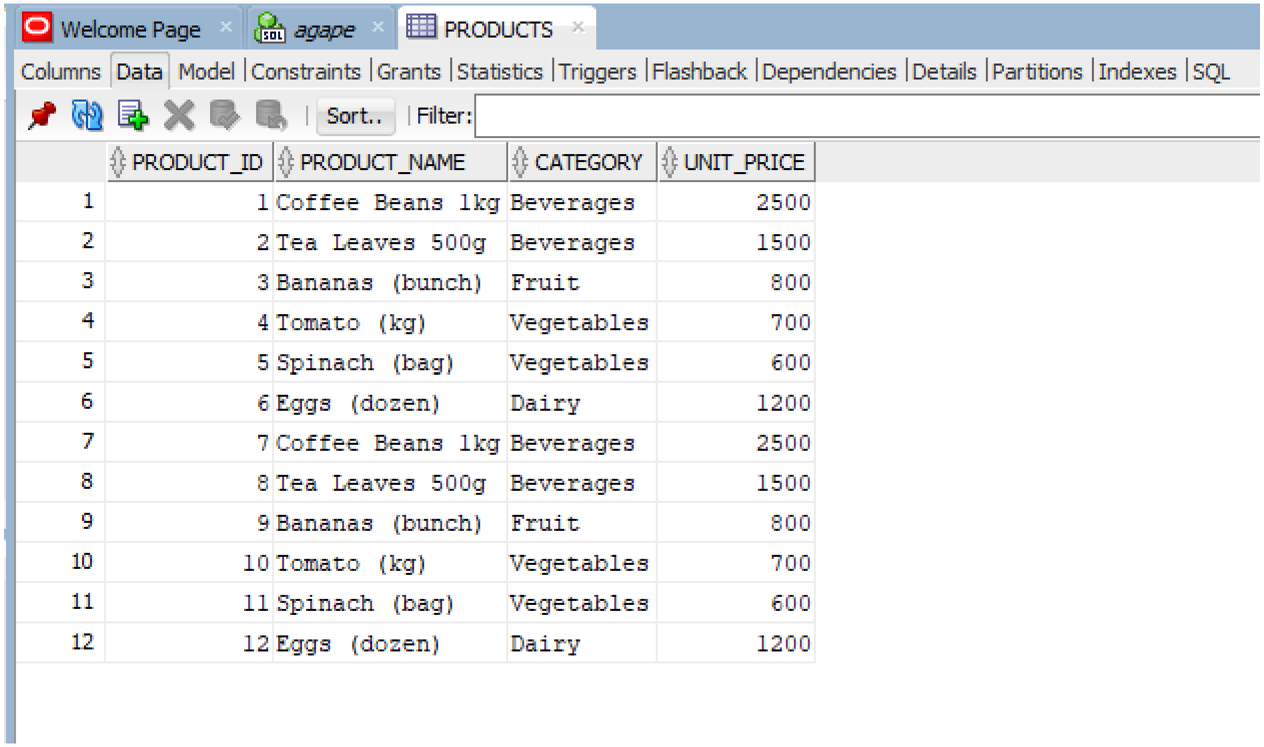
Script Output
Task completed in 9.879 seconds

```
Table PRODUCTS created.


Table ORDERS created.


Table ORDER_ITEMS created.
```

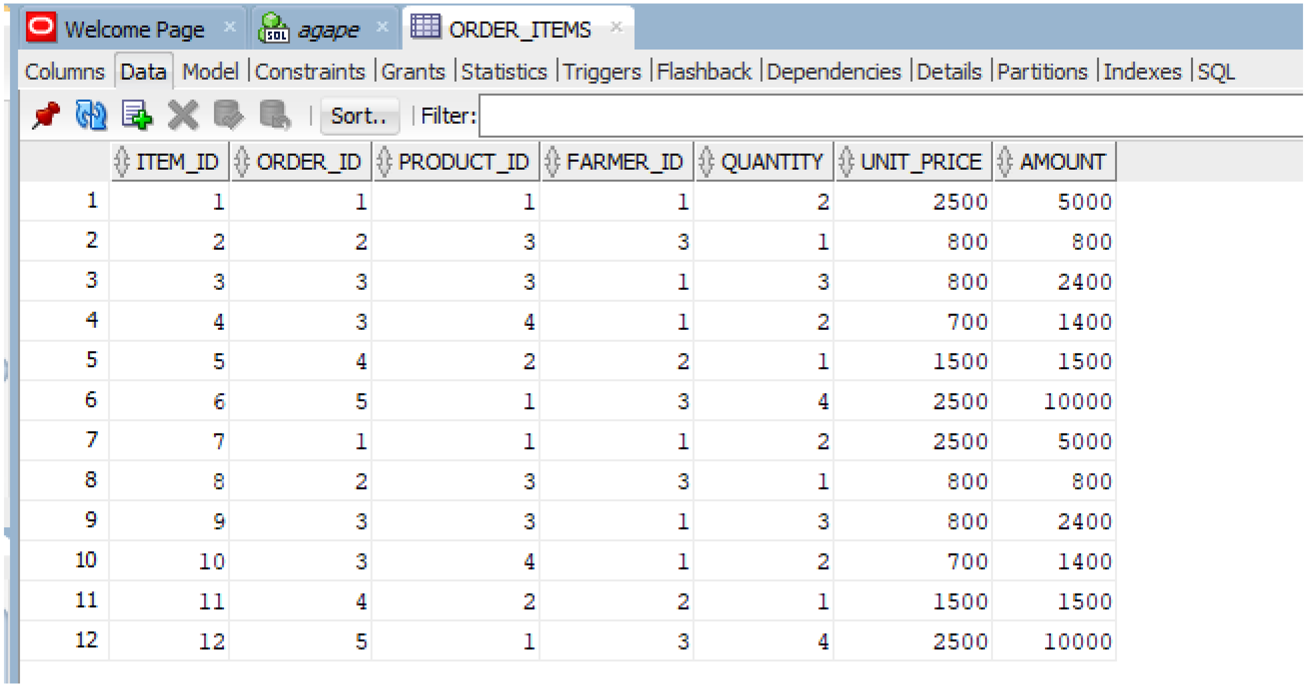
	REGION_ID	REGION_NAME
1	1	Kigali Central
2	2	Kigali East
3	3	Huye
4	4	Kigali Central
5	5	Kigali East
6	6	Huye

Script Output
Task completed in 37.94 seconds

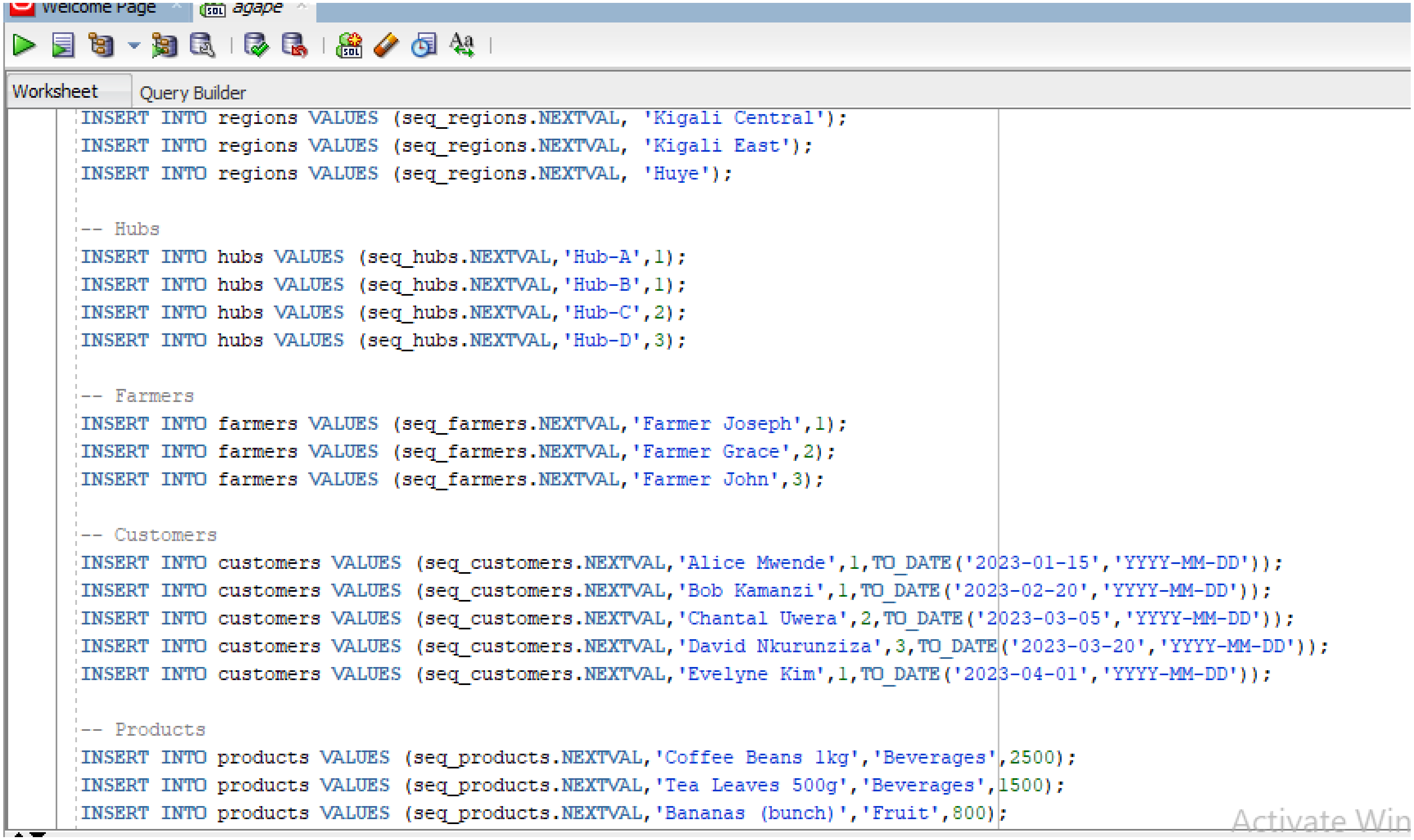
```
Table REGIONS created.


Table HUBS created.


Table FARMERS created.


Table CUSTOMERS created.
```


Welcome Page
agape
PRODUCTS
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL
Sort..
Filter:

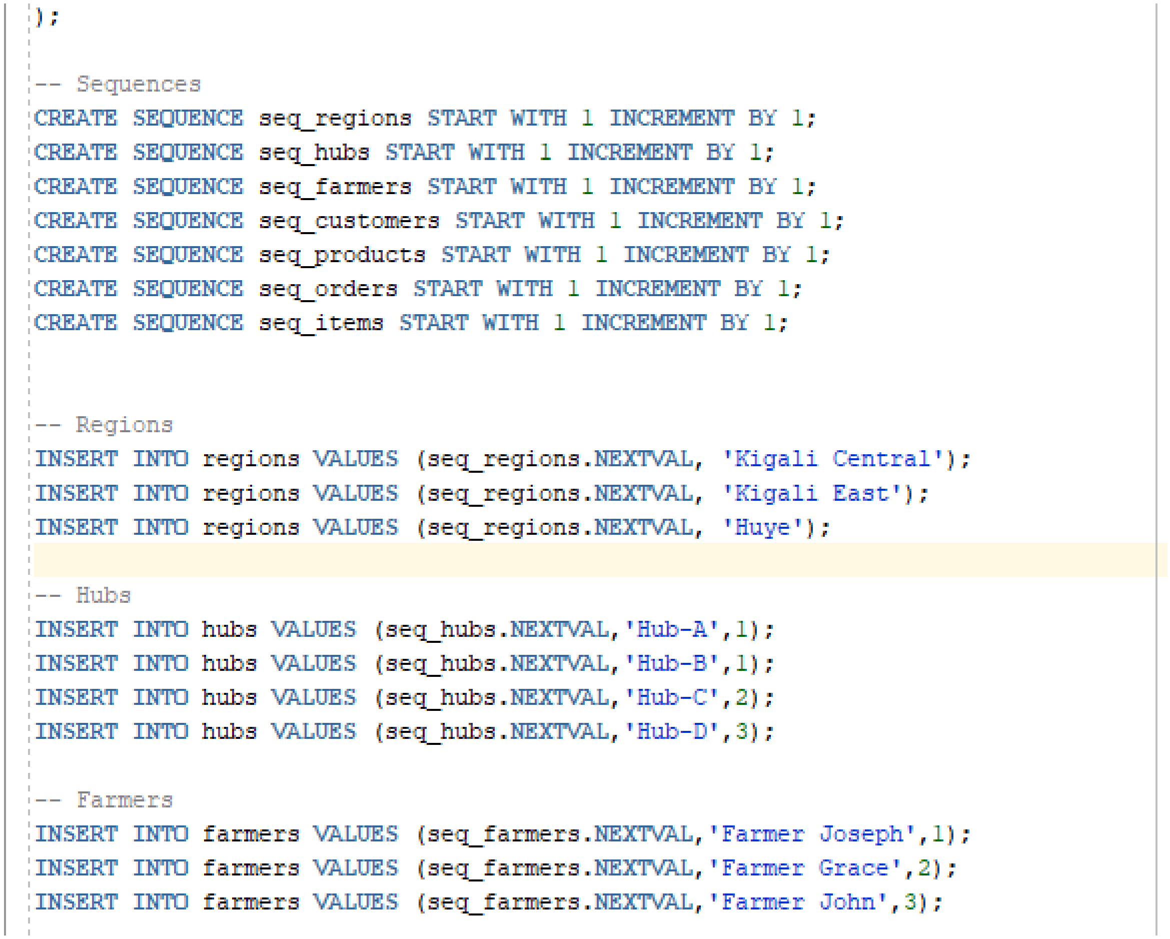
	PRODUCT_ID	PRODUCT_NAME	CATEGORY	UNIT_PRICE
1	1	Coffee Beans 1kg	Beverages	2500
2	2	Tea Leaves 500g	Beverages	1500
3	3	Bananas (bunch)	Fruit	800
4	4	Tomato (kg)	Vegetables	700
5	5	Spinach (bag)	Vegetables	600
6	6	Eggs (dozen)	Dairy	1200
7	7	Coffee Beans 1kg	Beverages	2500
8	8	Tea Leaves 500g	Beverages	1500
9	9	Bananas (bunch)	Fruit	800
10	10	Tomato (kg)	Vegetables	700
11	11	Spinach (bag)	Vegetables	600
12	12	Eggs (dozen)	Dairy	1200



	ITEM_ID	ORDER_ID	PRODUCT_ID	FARMER_ID	QUANTITY	UNIT_PRICE	AMOUNT
1	1	1	1	1	2	2500	5000
2	2	2	3	3	1	800	800
3	3	3	3	1	3	800	2400
4	4	3	4	1	2	700	1400
5	5	4	2	2	1	1500	1500
6	6	5	1	3	4	2500	10000
7	7	1	1	1	2	2500	5000
8	8	2	3	3	1	800	800
9	9	3	3	1	3	800	2400
10	10	3	4	1	2	700	1400
11	11	4	2	2	1	1500	1500
12	12	5	1	3	4	2500	10000



```sql
INSERT INTO regions VALUES (seq_regions.NEXTVAL, 'Kigali Central');
INSERT INTO regions VALUES (seq_regions.NEXTVAL, 'Kigali East');
INSERT INTO regions VALUES (seq_regions.NEXTVAL, 'Huye');

-- Hubs
INSERT INTO hubs VALUES (seq_hubs.NEXTVAL,'Hub-A',1);
INSERT INTO hubs VALUES (seq_hubs.NEXTVAL,'Hub-B',1);
INSERT INTO hubs VALUES (seq_hubs.NEXTVAL,'Hub-C',2);
INSERT INTO hubs VALUES (seq_hubs.NEXTVAL,'Hub-D',3);

-- Farmers
INSERT INTO farmers VALUES (seq_farmers.NEXTVAL,'Farmer Joseph',1);
INSERT INTO farmers VALUES (seq_farmers.NEXTVAL,'Farmer Grace',2);
INSERT INTO farmers VALUES (seq_farmers.NEXTVAL,'Farmer John',3);

-- Customers
INSERT INTO customers VALUES (seq_customers.NEXTVAL,'Alice Mwende',1,TO_DATE('2023-01-15','YYYY-MM-DD'));
INSERT INTO customers VALUES (seq_customers.NEXTVAL,'Bob Kamanzi',1,TO_DATE('2023-02-20','YYYY-MM-DD'));
INSERT INTO customers VALUES (seq_customers.NEXTVAL,'Chantal Uwera',2,TO_DATE('2023-03-05','YYYY-MM-DD'));
INSERT INTO customers VALUES (seq_customers.NEXTVAL,'David Nkurunziza',3,TO_DATE('2023-03-20','YYYY-MM-DD'));
INSERT INTO customers VALUES (seq_customers.NEXTVAL,'Evelyne Kim',1,TO_DATE('2023-04-01','YYYY-MM-DD'));

-- Products
INSERT INTO products VALUES (seq_products.NEXTVAL,'Coffee Beans 1kg','Beverages',2500);
INSERT INTO products VALUES (seq_products.NEXTVAL,'Tea Leaves 500g','Beverages',1500);
INSERT INTO products VALUES (seq_products.NEXTVAL,'Bananas (bunch)','Fruit',800);
```



```
);

-- Sequences
CREATE SEQUENCE seq_regions START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_hubs START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_farmers START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_customers START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_products START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_orders START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_items START WITH 1 INCREMENT BY 1;


-- Regions
INSERT INTO regions VALUES (seq_regions.NEXTVAL, 'Kigali Central');
INSERT INTO regions VALUES (seq_regions.NEXTVAL, 'Kigali East');
INSERT INTO regions VALUES (seq_regions.NEXTVAL, 'Huye');

-- Hubs
INSERT INTO hubs VALUES (seq_hubs.NEXTVAL,'Hub-A',1);
INSERT INTO hubs VALUES (seq_hubs.NEXTVAL,'Hub-B',1);
INSERT INTO hubs VALUES (seq_hubs.NEXTVAL,'Hub-C',2);
INSERT INTO hubs VALUES (seq_hubs.NEXTVAL,'Hub-D',3);

-- Farmers
INSERT INTO farmers VALUES (seq_farmers.NEXTVAL,'Farmer Joseph',1);
INSERT INTO farmers VALUES (seq_farmers.NEXTVAL,'Farmer Grace',2);
INSERT INTO farmers VALUES (seq_farmers.NEXTVAL,'Farmer John',3);
```

6. Results Analysis

- **Descriptive Analysis:** We identified the best-selling products per hub and observed a strong overall sales growth, particularly in March.

Diagnostic Analysis: The month-over-month analysis diagnosed March as a period of exceptional growth, which merits further investigation to understand the contributing factors.

Prescriptive Analysis: We prescribe focusing loyalty efforts on top-quartile customers and using moving averages for reliable inventory forecasting to maintain operational efficiency.

7. References

1. Oracle 21c SQL Language Reference
2. VideoTutorials -- SQL Window Functions
3. Course Lecture Notes