PL/SQL Window Functions Project

Student Name: Nyarugabo Daniel 29192

Project Title: FreshMark Online Agricultural Marketplace

Q1. Problem Definition & Business Context

FreshMark is a website where people can buy things from farmers. FreshMark is like a store. The website is a place where farmers can put up their products for sale.

Business Challenges

It s difficult for farmers to find the customer and where to sell their product

Proposed Solution

FreshMark makes the agricultural marketplace easier to use by:

Connecting farmers directly with buyers

Increasing farmers' profits

Providing customers with fresh, affordable, locally produced food

Using data analytics to support business decisions

Q2. Success Criteria

Goal 1: Analyze sales performance by product category

Window function to use: RANK()

This function helps us see which products are doing well and which ones are not doing well in each category. It does this by looking at how sales each product has had. The function ranks the products in each category based on sales.

Goal 2: Track cumulative sales month by month

Window function to use: SUM() OVER()

The reason is that it lets the sales values add up over time. This helps the business keep an eye on how the overall revenue's growing from one month to another.

Goal 3: Measure sales growth compared to the previous month

Window function to use: LAG()

The reason for this function is that it looks at the sales for the month and the sales for the previous month. This helps us see if the sales have gone up or down.

Goal 4: Segment customers based on purchasing behavior

Window function to use: NTILE(4

The reason is that it separates the customers into four groups based on how much they spend. This helps with targeted marketing and customer retention strategies, for the customers.

Goal 5: I need to calculate the sales over three months. This is what we call a three-month moving average of sales. So we will add up the sales for three months. Then divide by three to get the average sales, for that period.

Window function to use: AVG() OVER()

Why: This function smooths short-term fluctuations in sales data and highlights long-term sales trends.

Q3. Database Schema Design

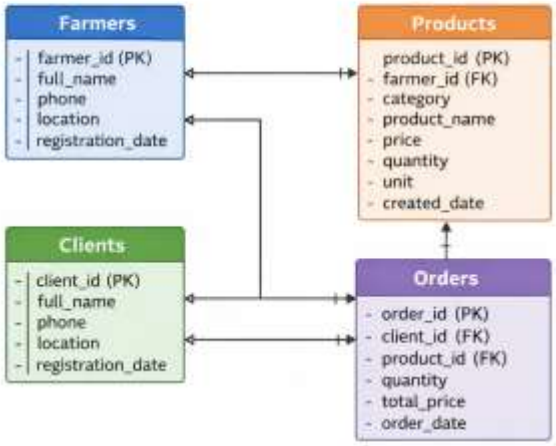


Figure 1: Database Schema

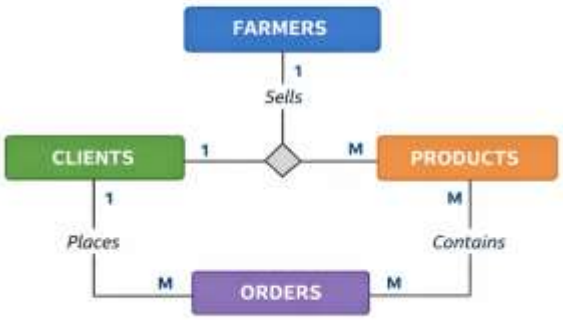
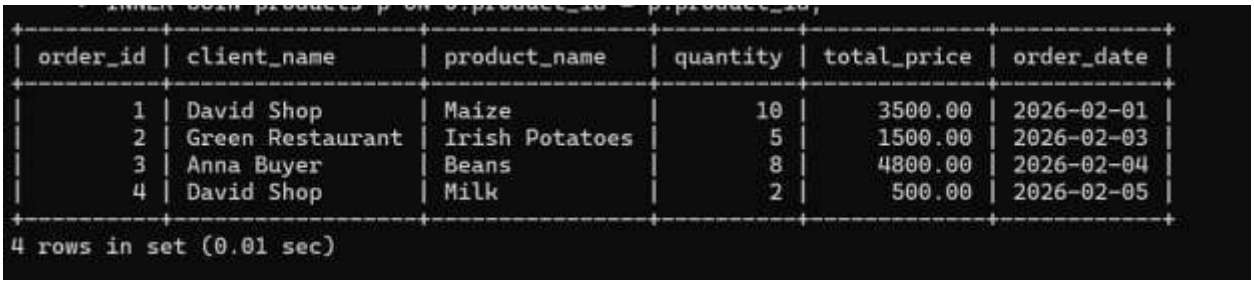


Figure 2: Entity Relationship Diagram (ERD)

Q4. SQL JOINs Implementation

1. INNER JOIN – Valid Orders

```
mysql> SELECT
    ->     o.order_id,
    ->     c.full_name AS client_name,
    ->     p.product_name,
    ->     o.quantity,
    ->     o.total_price,
    ->     o.order_date
    -> FROM orders o
    -> INNER JOIN clients c ON o.client_id = c.client_id
    -> INNER JOIN products p ON o.product_id = p.product_id;OUT PUT
```



order_id	client_name	product_name	quantity	total_price	order_date
1	David Shop	Maize	10	3500.00	2026-02-01
2	Green Restaurant	Irish Potatoes	5	1500.00	2026-02-03
3	Anna Buyer	Beans	8	4800.00	2026-02-04
4	David Shop	Milk	2	500.00	2026-02-05

4 rows in set (0.01 sec)

Business Interpretation:

This thing shows transactions that we know are real. Only when we have a client and we also have a product. It has to have both the client and the product to show the transaction.

2. LEFT JOIN – Clients With No Orders

mysql> SELECT

-> c.client_id,

-> c.full_name,

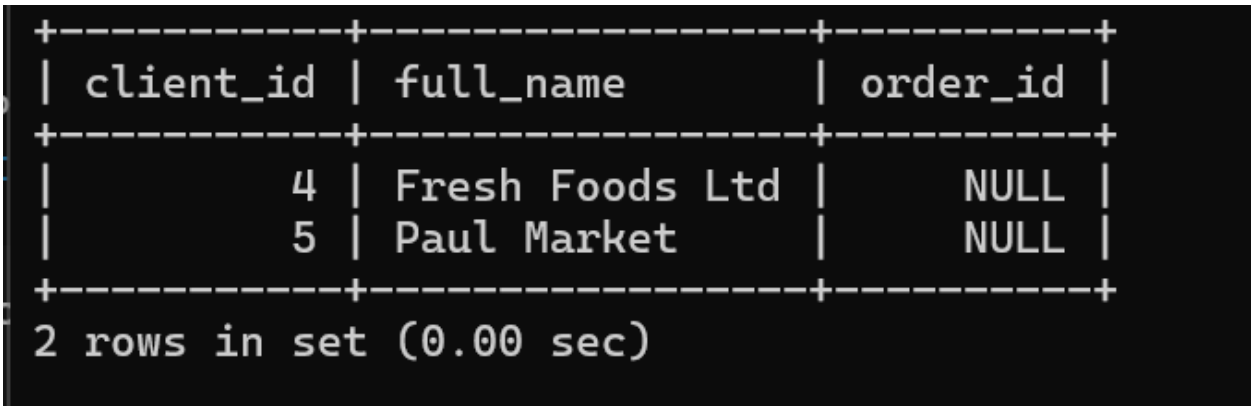
-> o.order_id

-> FROM clients c

-> LEFT JOIN orders o ON c.client_id = o.client_id

-> WHERE o.order_id IS NULL;

OUT OUT



client_id	full_name	order_id
4	Fresh Foods Ltd	NULL
5	Paul Market	NULL

2 rows in set (0.00 sec)

Business Interpretation:

Identifies inactive customers for promotions or engagement campaigns.

3. RIGHT JOIN – Products With No Sales

```
mysql> SELECT
    ->     p.product_id,
    ->     p.product_name,
    ->     o.order_id
    -> FROM orders o
    -> RIGHT JOIN products p ON o.product_id = p.product_id
    -> WHERE o.order_id IS NULL;OUTPUT
```

```
+------------+--------------+----------+
| product_id | product_name | order_id |
+------------+--------------+----------+
|          5 | Eggs         |     NULL |
+------------+--------------+----------+
1 row in set (0.00 sec)
```

Business Interpretation:

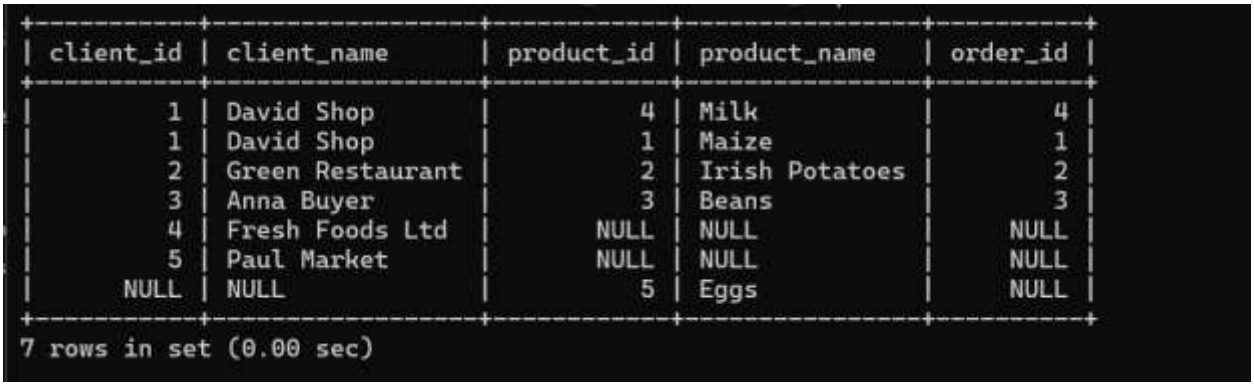
Helps detect unsold products for pricing or marketing review.

4. FULL OUTER JOIN – Complete Market Overview

```
SELECT
c.client_id,
c.full_name AS client_name,
p.product_id,
p.product_name,
o.order_id
FROM clients c
FULL OUTER JOIN orders o ON c.client_id = o.client_id
```

FULL OUTER JOIN products p ON o.product_id = p.product_id;

OUTPUT



client_id	client_name	product_id	product_name	order_id
1	David Shop	4	Milk	4
1	David Shop	1	Maize	1
2	Green Restaurant	2	Irish Potatoes	2
3	Anna Buyer	3	Beans	3
4	Fresh Foods Ltd	NULL	NULL	NULL
5	Paul Market	NULL	NULL	NULL
NULL	NULL	5	Eggs	NULL

7 rows in set (0.00 sec)

Business Interpretation:

Shows all customers and products, including unmatched records.

5. SELF JOIN – Clients in the Same Location

SELECT

c1.client_id AS client1_id,

c1.full_name AS client1_name,

c2.client_id AS client2_id,

c2.full_name AS client2_name,

c1.location

FROM clients c1

JOIN clients c2

ON c1.location = c2.location

AND c1.client_id < c2.client_id;

OUT OUT

```
+------------+--------------+------------+-----------------+----------+
| client1_id | client1_name | client2_id | client2_name    | location |
+------------+--------------+------------+-----------------+----------+
|          1 | David Shop   |          2 | Green Restaurant | Kigali   |
+------------+--------------+------------+-----------------+----------+
1 row in set (0.00 sec)
```

Business Interpretation:

Finds pairs of clients located in the same city. Useful for targeting region specific marketing or logics planning.

Q5. Window functions

1. RANK()

```
mysql> SELECT
    ->     p.category,
    ->     p.product_name,
    ->     SUM(o.total_price) AS total_sales,
    ->     RANK() OVER (
    ->         PARTITION BY p.category
    ->         ORDER BY SUM(o.total_price) DESC
    ->     ) AS sales_rank
    -> FROM products p
    -> JOIN orders o ON p.product_id = o.product_id
    -> GROUP BY p.category, p.product_name;
+------------+----------------+-------------+------------+
| category   | product_name   | total_sales | sales_rank |
+------------+----------------+-------------+------------+
| Dairy      | Milk           |      500.00 |          1 |
| Grains     | Maize          |     3500.00 |          1 |
| Legumes    | Beans          |     4800.00 |          1 |
| Vegetables | Irish Potatoes |     1500.00 |          1 |
+------------+----------------+-------------+------------+
4 rows in set (0.01 sec)
```

2. SUM() AND OVER

```
mysql> SELECT
    ->     t.month,
    ->     t.monthly_sales,
    ->     SUM(t.monthly_sales) OVER (ORDER BY t.month) AS cumulative_sales
    -> FROM (
    ->     SELECT
    ->         DATE_FORMAT(order_date, '%Y-%m') AS month,
    ->         SUM(total_price) AS monthly_sales
    ->     FROM orders
    ->     GROUP BY DATE_FORMAT(order_date, '%Y-%m')
    -> ) AS t;
+---------+---------------+------------------+
| month   | monthly_sales | cumulative_sales |
+---------+---------------+------------------+
| 2026-02 |      10300.00 |         10300.00 |
+---------+---------------+------------------+
1 row in set (0.00 sec)
```

3. LAG()

```
mysql> SELECT
    ->     t.month,
    ->     t.monthly_sales,
    ->     LAG(t.monthly_sales) OVER (ORDER BY t.month) AS
    -> FROM (
    ->     SELECT
    ->         DATE_FORMAT(order_date, '%Y-%m') AS month,
    ->         SUM(total_price) AS monthly_sales
    ->     FROM orders
    ->     GROUP BY DATE_FORMAT(order_date, '%Y-%m')
    -> ) AS t;
+---------+---------------+----------------------+
| month   | monthly_sales | previous_month_sales |
+---------+---------------+----------------------+
| 2026-02 |      10300.00 |                 NULL |
+---------+---------------+----------------------+
1 row in set (0.00 sec)
```

4. NTILE()

```
mysql> SELECT
    ->     c.client_id,
    ->     c.full_name,
    ->     SUM(o.total_price) AS total_spent,
    ->     NTILE(4) OVER (ORDER BY SUM(o.total_price) DESC) AS customer_group
    -> FROM clients c
    -> JOIN orders o ON c.client_id = o.client_id
    -> GROUP BY c.client_id, c.full_name;
+-----------+----------------+-------------+----------------+
| client_id | full_name      | total_spent | customer_group |
+-----------+----------------+-------------+----------------+
|         3 | Anna Buyer     |     4800.00 |              1 |
|         1 | David Shop     |     4000.00 |              2 |
|         2 | Green Restaurant |     1500.00 |              3 |
+-----------+----------------+-------------+----------------+
3 rows in set (0.00 sec)
```

5. AVG(), OVER()

```
mysql> SELECT
    ->     t.month,
    ->     t.monthly_sales,
    ->     AVG(t.monthly_sales) OVER (
    ->         ORDER BY t.month
    ->         ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ->     ) AS three_month_avg
    -> FROM (
    ->     SELECT
    ->         DATE_FORMAT(order_date, '%Y-%m') AS month,
    ->         SUM(total_price) AS monthly_sales
    ->     FROM orders
    ->     GROUP BY DATE_FORMAT(order_date, '%Y-%m')
    -> ) AS t;
+---------+---------------+-----------------+
| month   | monthly_sales | three_month_avg |
+---------+---------------+-----------------+
| 2026-02 |      10300.00 |    10300.000000 |
+---------+---------------+-----------------+
1 row in set (0.00 sec)
```