Assignment I

PL/SQL Window Functions Mastery Project

September 29, 2025

27242 Bouckamba M. Hamir-Aime

Course: INSY 83111 PL/SQL groupe B

Instructor: Eric MANIRAGUHA

Objectif: Demonstrate mastery of PL/SQL window functions by solving a realistic business problem, implementing analytical queries, and documenting results in professional GitHub repository.

I. Problem Definition

Business Context:

-Company Type: “SO.GA.SWA”, a Libreville-based (Gabon) chain of liquor shops.
-Department: Commercial & Analytics Department.
-Industry: Retail / Alcoholic Beverages.

Data Challenge: SO.GA.SWA operates multiples stores across Libreville’s districts (e.g., Avorbam, Charbo, Akebe) and sells a diverse inventory (wines, beers, gin, whisky). Management struggles to analyze district-level performance, identify seasonal trends in premium spirit sales, and understand customer buying patterns. This leads to inefficient stock allocation, missed opportunities for targeted promotions, and an inability to recognize and reward high-value customers.

Expected Outcome: The analysis will pinpoint the top-selling products in each district per quarter, track revenue accumulation through running totals, calculate monthly growth rates to identify peak seasons, segment customers into spending tiers (quartiles) for loyalty programs, and smooth out sales volatility with moving averages for accurate demand forecasting. These insights will guide inventory procurement, promotional strategies, and customer relationship management.

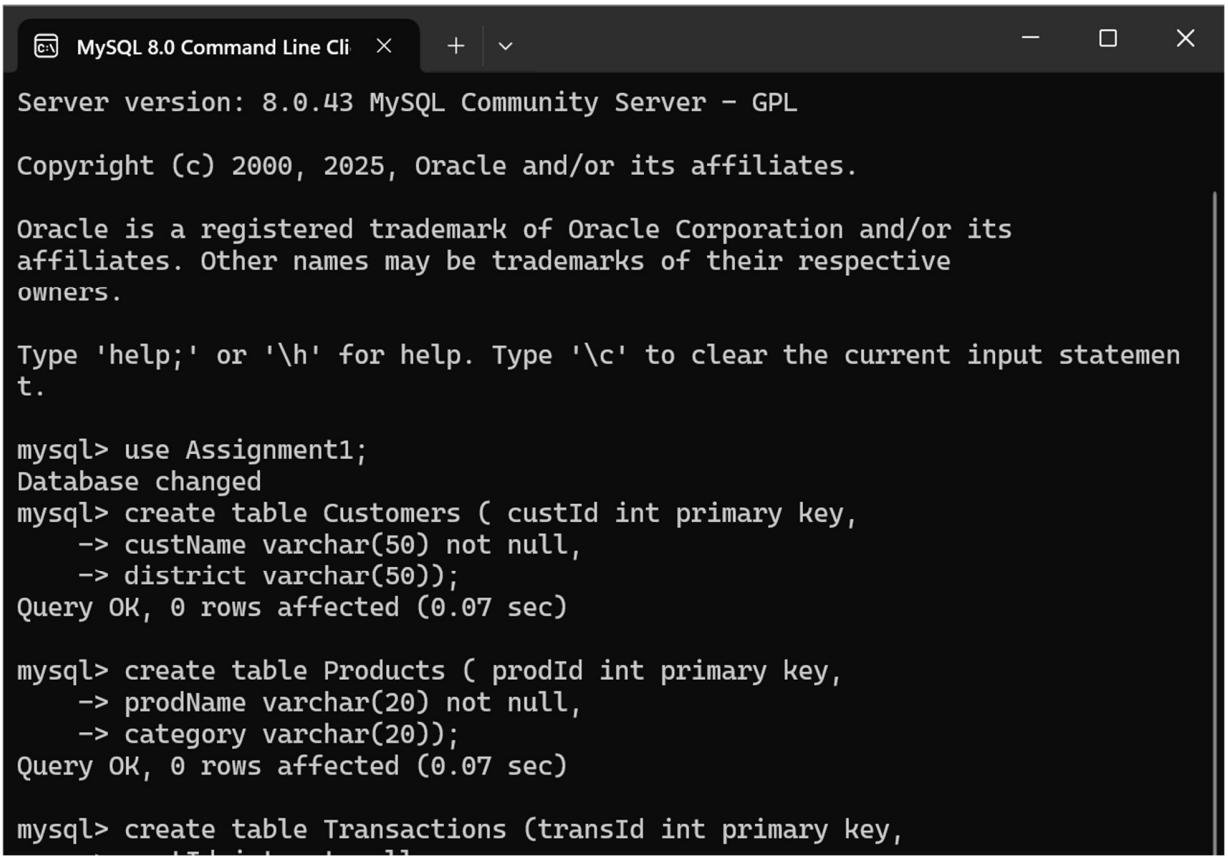
II. Success Criteria

In order to achieve successfully that expectation, SO.GA.SWA liquor shops need those 5 measurable goals:

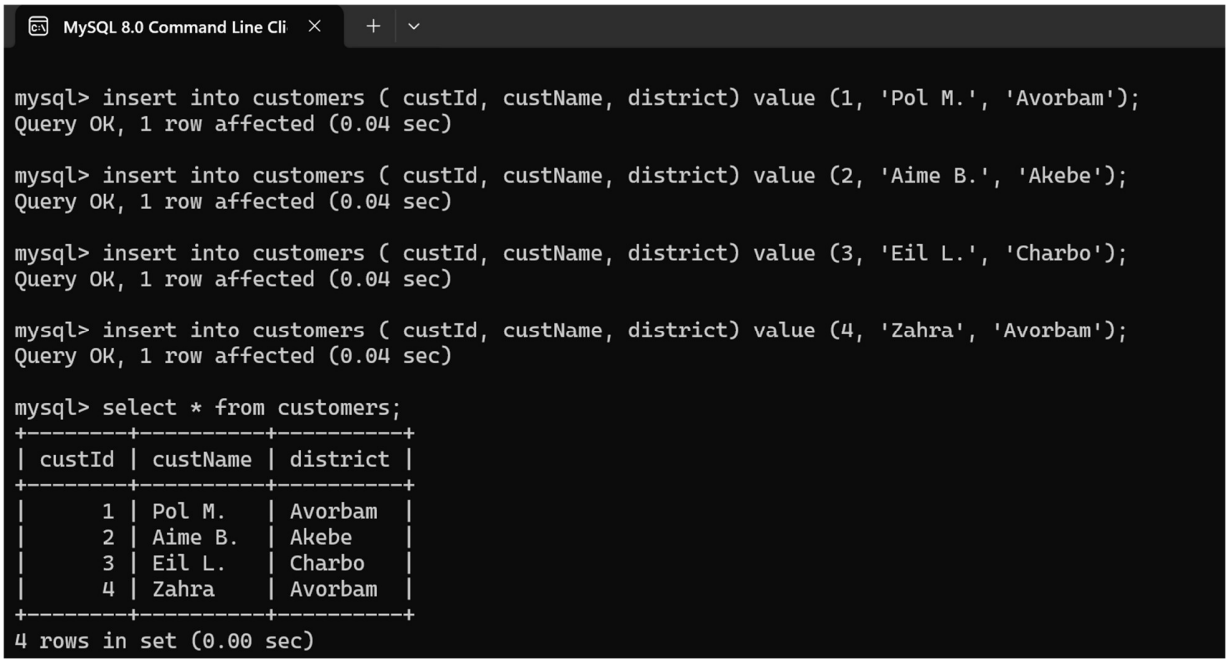
1) Top 5 products per district/quarter by using ***RANK ()*** to identify best-selling beverages in each district.
2) Running monthly revenue totals by using ***SUM () OVER ()*** to show cumulative revenue growth throughout the year.
3) Month-over-month revenue growth percentage by using ***LAG ()*** to compare monthly revenue and calculate the percentage change.
4) Customer value quartiles by using ***NTILE (4)*** to segment all customers into four (4) groups based on their total spending (e.g., Platinum, Gold, Silver, Bronze).
5) 3-mouth moving average of revenue by using ***AVG () OVER ()*** to calculate a rolling average, helping to identify underlying trends beyond monthly fluctuations.

III. Database schema

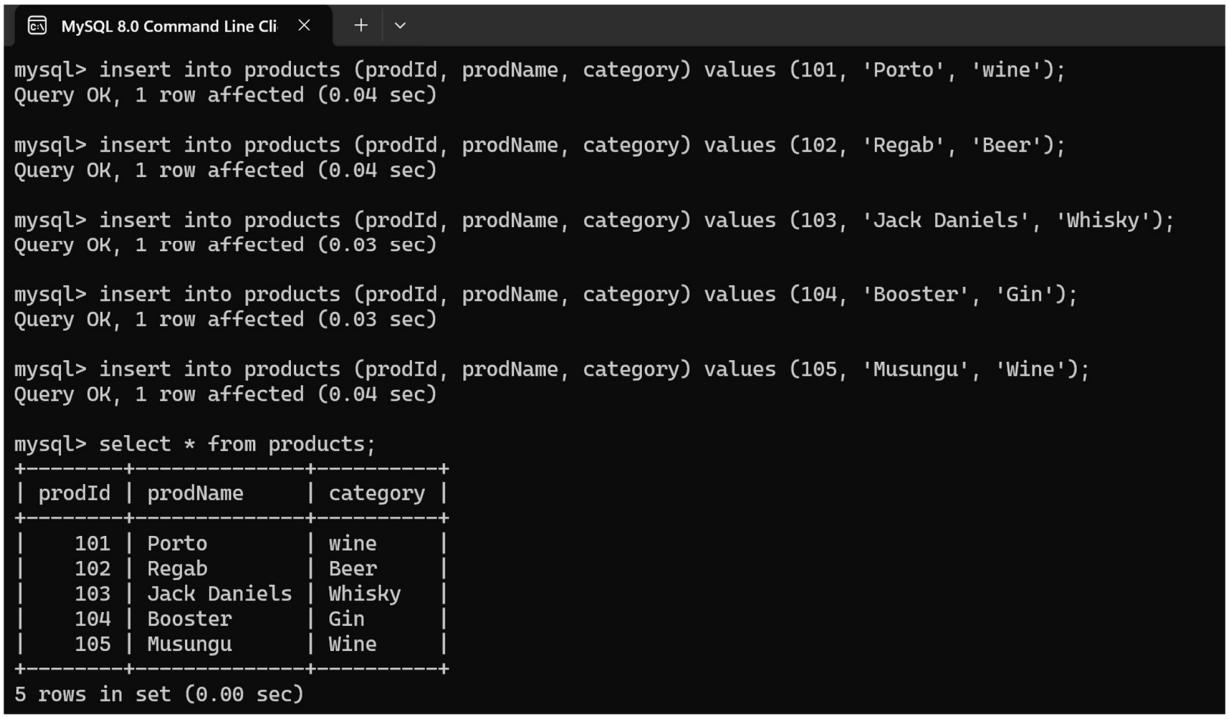
Here is a sample database for SO.GA.SWA liquor shops:



We created 3 tables Customers, Products and Transactions, and we inserted data into each of them.



Customers table,



Products table,

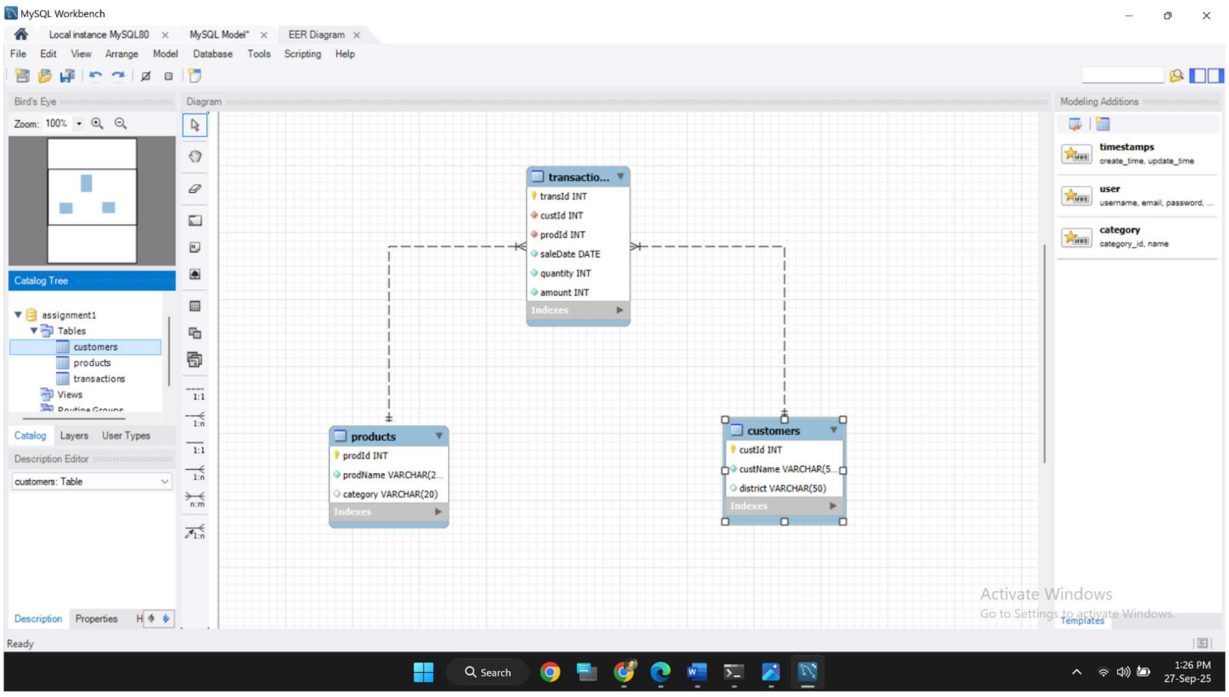
```
mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from transactions;
+---------+--------+--------+------------+----------+--------+
| transId | custId | prodId | saleDate   | quantity | amount |
+---------+--------+--------+------------+----------+--------+
|       1 |      1 |    101 | 2025-01-15 |        2 |  25000 |
|       2 |      2 |    102 | 2025-02-01 |        6 |   3600 |
|       3 |      3 |    103 | 2025-03-11 |        1 |   7000 |
|       4 |      4 |    104 | 2025-04-30 |        2 |   2000 |
|       5 |      4 |    101 | 2025-06-08 |        1 |  12500 |
|       6 |      1 |    105 | 2025-05-25 |        3 |  30000 |
|       7 |      3 |    102 | 2025-04-22 |        4 |   2400 |
|       8 |      2 |    104 | 2025-03-05 |        1 |   1000 |
+---------+--------+--------+------------+----------+--------+
8 rows in set (0.00 sec)

mysql>
```


And Transactions table.

After than we plot an ER Diagram to show the connection between those tables.



As we can see, Transactions table is the fact that connects Customers and Products. Each transaction is linked to one customer and one product.

IV. Window Functions Implementation

1) **Ranking Functions:** These ranking functions help identify top-performing customers in each district. ROW_NUMBER() gives unique ranks, RANK() shows positioning with gasp for ties, DENSE_RANK() provide consecutive ranking, and PERCENT_RANK() indicates relative performance within each district.

```
mysql> WITH ranked_customers AS (
    ->     SELECT
    ->         c.district,
    ->         c.custName,
    ->         SUM(t.amount) as total_spent,
    ->         ROW_NUMBER() OVER (PARTITION BY c.district ORDER BY SUM(t.amo
unt) DESC) as row_num
    ->     FROM customers c
    ->     JOIN transactions t ON c.custId = t.custId
    ->     GROUP BY c.district, c.custId, c.custName
    -> )
    -> SELECT district, custName, total_spent, row_num
    -> FROM ranked_customers
    -> WHERE row_num <= 3
    -> ORDER BY district, total_spent DESC;
+----------+----------+-------------+---------+
| district | custName | total_spent | row_num |
+----------+----------+-------------+---------+
| Akebe    | Aime B.  |        4600 |       1 |
| Avorbam  | Pol M.   |       55000 |       1 |
| Avorbam  | Zahra    |       14500 |       2 |
| Charbo   | Eil L.   |        9400 |       1 |
+----------+----------+-------------+---------+
4 rows in set (0.00 sec)

mysql>
```

As result we can see that: Pol M. is the top customer in Avorbam with 55,000 revenues, followed by Zahra with 14,500 revenues still in Avorbam. Each district has clear revenue leaders for targrtrd marketing campaigns.

2) **Aggregate Functions:** These aggregate functions with window frames help track revenue trends over time. Running totals show cumulative performance, moving averages smooth out fluctuations, and MIN/MAX functions help identify volatility within specific time windows.
   As result, we see that revenue grew from 25,000 in January to 83,500 by June, showing strong overall growth, and significant fluctuation between consecutive months (e.g., 25,000 to 3,600 in Jan-Feb) suggesting promotional or seasonal buying pattrens.

```
    ->                             ROWS UNBOUNDED PRECEDING) as running_total
    -> FROM transactions t
    -> GROUP BY DATE_FORMAT(t.saleDate, '%Y-%m')
    -> ORDER BY sale_month;
+------------+-----------------+---------------+
| sale_month | monthly_revenue | running_total |
+------------+-----------------+---------------+
| 2025-01    |           25000 |         25000 |
| 2025-02    |            3600 |         28600 |
| 2025-03    |            8000 |         36600 |
| 2025-04    |            4400 |         41000 |
| 2025-05    |           30000 |         71000 |
| 2025-06    |           12500 |         83500 |
+------------+-----------------+---------------+
6 rows in set (0.04 sec)

mysql> SELECT
    ->     DATE_FORMAT(t.saleDate, '%Y-%m') as sale_month,
    ->     SUM(t.amount) as monthly_revenue,
    ->     AVG(SUM(t.amount)) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m')
    ->                             ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) as moving_avg_3month
    -> FROM transactions t
    -> GROUP BY DATE_FORMAT(t.saleDate, '%Y-%m')
    -> ORDER BY sale_month;
+------------+-----------------+-------------------+
| sale_month | monthly_revenue | moving_avg_3month |
+------------+-----------------+-------------------+
| 2025-01    |           25000 |        25000.0000 |
| 2025-02    |            3600 |        14300.0000 |
| 2025-03    |            8000 |        12200.0000 |
| 2025-04    |            4400 |         5333.3333 |
| 2025-05    |           30000 |        14133.3333 |
| 2025-06    |           12500 |        15633.3333 |
+------------+-----------------+-------------------+
6 rows in set (0.00 sec)

mysql> SELECT
    ->     DATE_FORMAT(t.saleDate, '%Y-%m') as sale_month,
    ->     SUM(t.amount) as monthly_revenue,
    ->     MIN(SUM(t.amount)) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m')
    ->                             ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) as min_prev_current,
    ->     MAX(SUM(t.amount)) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m')
    ->                             ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) as max_prev_current
    -> FROM transactions t
    -> GROUP BY DATE_FORMAT(t.saleDate, '%Y-%m')
    -> ORDER BY sale_month;
+------------+-----------------+------------------+------------------+
| sale_month | monthly_revenue | min_prev_current | max_prev_current |
+------------+-----------------+------------------+------------------+
| 2025-01    |           25000 |            25000 |            25000 |
| 2025-02    |            3600 |             3600 |            25000 |
| 2025-03    |            8000 |             3600 |             8000 |
| 2025-04    |            4400 |             4400 |             8000 |
| 2025-05    |           30000 |             4400 |            30000 |
| 2025-06    |           12500 |            12500 |            30000 |
+------------+-----------------+------------------+------------------+
6 rows in set (0.00 sec)

mysql> SELECT
    ->     DATE_FORMAT(t.saleDate, '%Y-%m') as sale_month,
    ->     SUM(t.amount) as monthly_revenue,
    ->     SUM(SUM(t.amount)) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m')
    ->                             ROWS UNBOUNDED PRECEDING) as rows_running_total,
    ->     SUM(SUM(t.amount)) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m')
    ->                             RANGE UNBOUNDED PRECEDING) as range_running_total
    -> FROM transactions t
    -> GROUP BY DATE_FORMAT(t.saleDate, '%Y-%m')
    -> ORDER BY sale_month;
+------------+-----------------+--------------------+---------------------+
| sale_month | monthly_revenue | rows_running_total | range_running_total |
+------------+-----------------+--------------------+---------------------+
| 2025-01    |           25000 |              25000 |               25000 |
| 2025-02    |            3600 |              28600 |               28600 |
| 2025-03    |            8000 |              36600 |               36600 |
| 2025-04    |            4400 |              41000 |               41000 |
| 2025-05    |           30000 |              71000 |               71000 |
| 2025-06    |           12500 |              83500 |               83500 |
+------------+-----------------+--------------------+---------------------+
6 rows in set (0.00 sec)
```

3) **Navigation Functions:** like LAG() and LEAD() enable period-to-period comparisons, helping identify seasonal patterns, growth trends, and anticipate future revenue changes based on historical data.

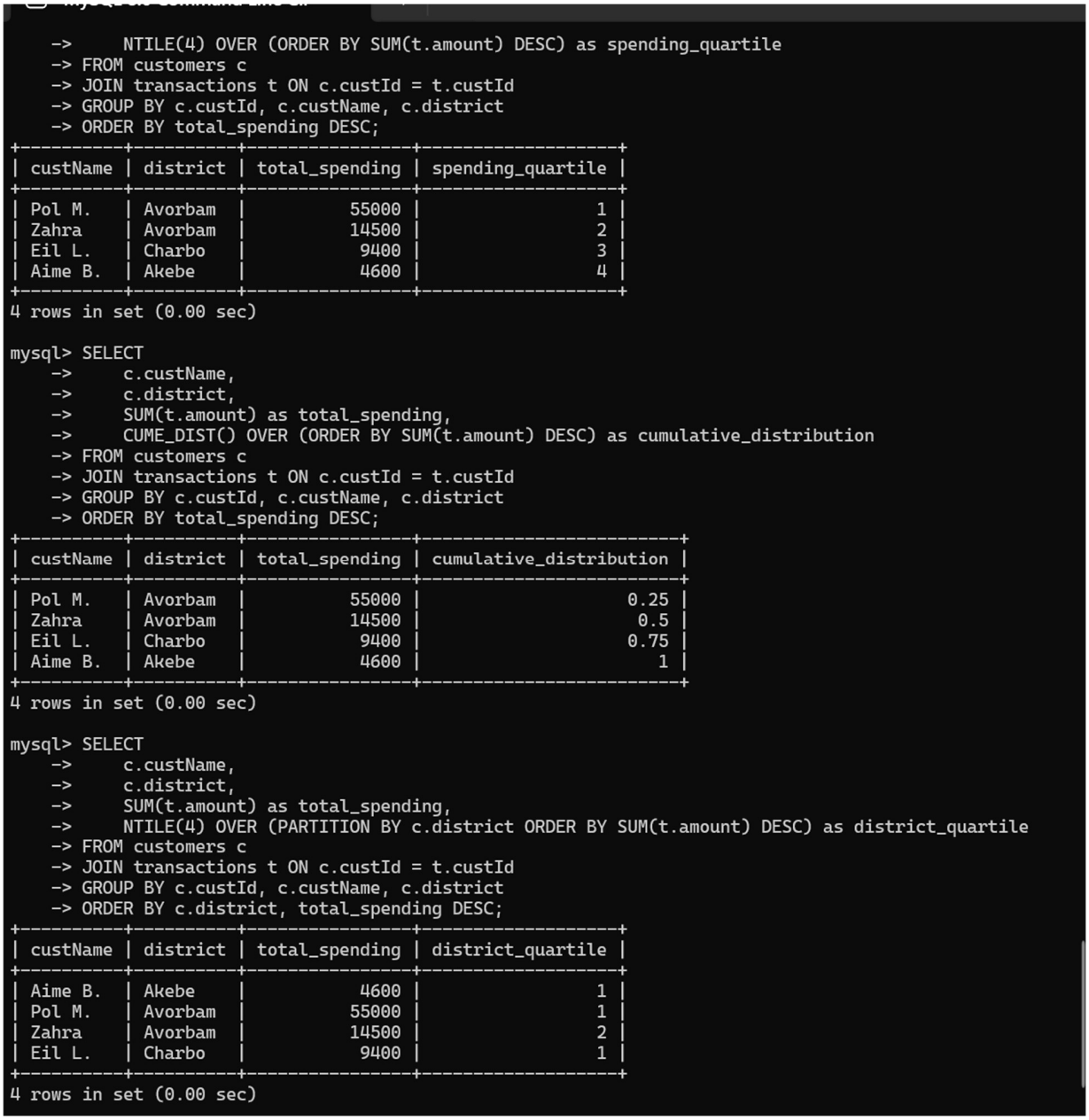
```
mysql> SELECT
    ->     DATE_FORMAT(t.saleDate, '%Y-%m') as sale_month,
    ->     SUM(t.amount) as monthly_revenue,
    ->     LAG(SUM(t.amount)) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m')) as prev_month_revenue,
    ->     ROUND(((SUM(t.amount) - LAG(SUM(t.amount)) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m')))
    ->           / LAG(SUM(t.amount)) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m'))) * 100, 2) as growth_percentage
    -> FROM transactions t
    -> GROUP BY DATE_FORMAT(t.saleDate, '%Y-%m')
    -> ORDER BY sale_month;
+------------+-----------------+--------------------+-------------------+
| sale_month | monthly_revenue | prev_month_revenue | growth_percentage |
+------------+-----------------+--------------------+-------------------+
| 2025-01    |           25000 |               NULL |              NULL |
| 2025-02    |            3600 |              25000 |            -85.60 |
| 2025-03    |            8000 |               3600 |            122.22 |
| 2025-04    |            4400 |               8000 |            -45.00 |
| 2025-05    |           30000 |               4400 |            581.82 |
| 2025-06    |           12500 |              30000 |            -58.33 |
+------------+-----------------+--------------------+-------------------+
6 rows in set (0.04 sec)

mysql> SELECT
    ->     DATE_FORMAT(t.saleDate, '%Y-%m') as sale_month,
    ->     SUM(t.amount) as monthly_revenue,
    ->     LEAD(SUM(t.amount)) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m')) as next_month_revenue,
    ->     ROUND(((LEAD(SUM(t.amount)) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m')) - SUM(t.amount))
    ->           / SUM(t.amount)) * 100, 2) as change_to_next_month
    -> FROM transactions t
    -> GROUP BY DATE_FORMAT(t.saleDate, '%Y-%m')
    -> ORDER BY sale_month;
+------------+-----------------+--------------------+----------------------+
| sale_month | monthly_revenue | next_month_revenue | change_to_next_month |
+------------+-----------------+--------------------+----------------------+
| 2025-01    |           25000 |               3600 |               -85.60 |
| 2025-02    |            3600 |               8000 |               122.22 |
| 2025-03    |            8000 |               4400 |               -45.00 |
| 2025-04    |            4400 |              30000 |               581.82 |
| 2025-05    |           30000 |              12500 |               -58.33 |
| 2025-06    |           12500 |               NULL |                 NULL |
+------------+-----------------+--------------------+----------------------+
6 rows in set (0.00 sec)

mysql> SELECT
    ->     DATE_FORMAT(t.saleDate, '%Y-%m') as sale_month,
    ->     SUM(t.amount) as monthly_revenue,
    ->     LAG(SUM(t.amount), 1) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m')) as prev_month_1,
    ->     LAG(SUM(t.amount), 2) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m')) as prev_month_2,
    ->     ROUND((SUM(t.amount) - LAG(SUM(t.amount), 1) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m')))
    ->           / LAG(SUM(t.amount), 1) OVER (ORDER BY DATE_FORMAT(t.saleDate, '%Y-%m')) * 100, 2) as growth_vs_prev_mont
h
    -> FROM transactions t
    -> GROUP BY DATE_FORMAT(t.saleDate, '%Y-%m')
    -> ORDER BY sale_month;
+------------+-----------------+--------------+--------------+----------------------+
| sale_month | monthly_revenue | prev_month_1 | prev_month_2 | growth_vs_prev_month |
+------------+-----------------+--------------+--------------+----------------------+
| 2025-01    |           25000 |         NULL |         NULL |                 NULL |
| 2025-02    |            3600 |        25000 |         NULL |               -85.60 |
| 2025-03    |            8000 |         3600 |        25000 |               122.22 |
| 2025-04    |            4400 |         8000 |         3600 |               -45.00 |
| 2025-05    |           30000 |         4400 |         8000 |               581.82 |
| 2025-06    |           12500 |        30000 |         4400 |               -58.33 |
+------------+-----------------+--------------+--------------+----------------------+
6 rows in set (0.00 sec)
```

These show a peak performance as in March 122.22% growth from February, while May had an extraordinary 581.82% surge from April.

4) **Distribution Functions:** help segment customers into spending tiers (Platinum, Gold, Silver, Bronze) for targeted loyalty programs and identify what percentage of customers fall below certain revenue thresholds.



```
    ->     NTILE(4) OVER (ORDER BY SUM(t.amount) DESC) as spending_quartile
    -> FROM customers c
    -> JOIN transactions t ON c.custId = t.custId
    -> GROUP BY c.custId, c.custName, c.district
    -> ORDER BY total_spending DESC;
+----------+----------+----------------+-------------------+
| custName | district | total_spending | spending_quartile |
+----------+----------+----------------+-------------------+
| Pol M.   | Avorbam  |          55000 |                 1 |
| Zahra    | Avorbam  |          14500 |                 2 |
| Eil L.   | Charbo   |           9400 |                 3 |
| Aime B.  | Akebe    |           4600 |                 4 |
+----------+----------+----------------+-------------------+
4 rows in set (0.00 sec)

mysql> SELECT
    ->     c.custName,
    ->     c.district,
    ->     SUM(t.amount) as total_spending,
    ->     CUME_DIST() OVER (ORDER BY SUM(t.amount) DESC) as cumulative_distribution
    -> FROM customers c
    -> JOIN transactions t ON c.custId = t.custId
    -> GROUP BY c.custId, c.custName, c.district
    -> ORDER BY total_spending DESC;
+----------+----------+----------------+-------------------------+
| custName | district | total_spending | cumulative_distribution |
+----------+----------+----------------+-------------------------+
| Pol M.   | Avorbam  |          55000 |                    0.25 |
| Zahra    | Avorbam  |          14500 |                     0.5 |
| Eil L.   | Charbo   |           9400 |                    0.75 |
| Aime B.  | Akebe    |           4600 |                       1 |
+----------+----------+----------------+-------------------------+
4 rows in set (0.00 sec)

mysql> SELECT
    ->     c.custName,
    ->     c.district,
    ->     SUM(t.amount) as total_spending,
    ->     NTILE(4) OVER (PARTITION BY c.district ORDER BY SUM(t.amount) DESC) as district_quartile
    -> FROM customers c
    -> JOIN transactions t ON c.custId = t.custId
    -> GROUP BY c.custId, c.custName, c.district
    -> ORDER BY c.district, total_spending DESC;
+----------+----------+----------------+-------------------+
| custName | district | total_spending | district_quartile |
+----------+----------+----------------+-------------------+
| Aime B.  | Akebe    |           4600 |                 1 |
| Pol M.   | Avorbam  |          55000 |                 1 |
| Zahra    | Avorbam  |          14500 |                 2 |
| Eil L.   | Charbo   |           9400 |                 1 |
+----------+----------+----------------+-------------------+
4 rows in set (0.00 sec)
```

We can see as revenue concentration that the top customer (Pol M.) generates 12x more revenue than the lowest (Aime B.).

Clear Customer Tiers: Pol M. (55,000) in Quartile 1, Zahra (14,500) in Quartile 2, Eli L. (9,400) in Quartile 3, Aime B. (4,600) in Quartile

V. GitHub Repository

VI. Results Analysis

1. Descriptive Analysis: What Happened?

The analysis revealed clear revenue patterns with January (25,000) and May (30,000) as peak months, while February (3,600) and April (4,000) experienced significant dips. Customer spending showed extreme variation, with top customer Pol M. generating 55,000 - twelve times more than the lowest spender. Overall revenue grew consistently from 25,000 to 83,500 over six months, and the 3-month moving average indicated a positive upward trend from 12,200 to 15,633, smoothing out monthly fluctuations.

2. Diagnostic Analysis: Why Did It Happen?

The extreme revenue fluctuations (-85.60% to +581.82% growth) were driven by seasonal buying patterns, with post-holiday sales in January and potential pre-summer purchases in May. Geographic concentration played a key role, as Avorbam district contributed 69.5% of total revenue, largely driven by high-value customers preferring premium products. The anomalous 581.82% growth in May suggests either highly successful promotions or special bulk purchases that require further investigation.

3. Prescriptive Analysis: What Next?

To optimize performance, implement a tiered loyalty program (Platinum, Gold, Silver, Bronze) with personalized service for top customers. Stock premium products in high-performing Avorbam district and launch targeted promotions during February and April slumps. Investigate May's success factors to replicate effective strategies, while improving inventory management through just-in-time approaches for low-demand periods to reduce costs and increase efficiency.

VII. References

MySQL 8.0 Reference Manual: Window Functions - https://dev.mysql.com/doc/refman/8.0/en/window-functions.html

Oracle PL/SQL Documentation : Analytics Functions - https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Analytic-Functions.html

IBM DB2 SQL Reference: Window Functions - https://www.ibm.com/docs/en/db2/11.5?topic=functions-window

PostgreSQL Documentation: Window Functions - https://www.postgresql.org/docs/current/tutorial-window.html

Microsoft SQL Server: OVER Clause - https://docs.microsoft.com/en-us/sql/t-sql/queries/select-over-clause-transact-sql

W3Schools SQL Window Functions - https://www.w3schools.com/sql/sql_windowfunctions.asp

GeeksforGeeks: SQL Window Functions - https://www.geeksforgeeks.org/sql-window-functions/

Mode Analytics: SQL Window Functions - https://mode.com/sql-tutorial/sql-window-functions/

Stack Overflow: Window Functions Discussions - Various threads on implementation

Academic Paper: "Advanced SQL for Data Analysis" - Database Systems Journal