PL/SQL Window Functions Assignment

NAME: NKURANGA BAZIGA Caleb ID: 28845

Table of Contents

Step 1: Problem Definition

Business Context: NBCaleb Hotel has a restaurant. People buy food, drinks, and cakes there.

Problem: The hotel writes down what people buy, but they do not know:

- Which food people like most
- How money goes up or down each month
- Which people spend a lot or a little

This makes it hard to plan and help customers well.

Expected Outcome: Find which food is liked most, see how money changes each month, know which customers spend a lot, and watch sales to make good choices.

Step 2: Success Criteria - 5 Measurable Goals

(a) Top 5 Products per Region/Quarter → RANK()

We look at which products earned the most money from our customers in each city or quarter.

Function: RANK()

How it works: It gives 1, 2, 3... to products based on money earned.

Our Data Outcome Example:

- Pizza Margherita → Rank 1 (lots of customers like Michael and Emma bought it)
- Sushi Roll → Rank 2 (bought by Sophia and James)
- Grilled Chicken → Rank 3 (local customers like John and Eric bought it)

Pizza made the most money, Sushi second, Chicken third.

(b) Running Monthly Sales Totals → SUM() OVER()

We check how much money we earned each month and keep adding it to see total so far.

Function: SUM() OVER()

Our Data Outcome Example:

- January = 5000 + 1000 + 3000 + ... = 23,000
- February = 5000 + 2000 + ... = 15,000
- Running total by Feb = 23,000 + 15,000 = 38,000

We earned 23,000 in Jan, 15,000 more in Feb, total so far = 38,000.

(c) Month-over-Month Growth → LAG()

We want to see if sales grew or dropped each month.

Function: LAG()

Our Data Outcome Example:

- January sales = 23,000, February = 15,000 → growth = -8,000
- March sales = 20,000 → growth = +5,000

Feb made less money than Jan, Mar made more money than Feb.

(d) Customer Quartiles → NTILE(4)

We split customers into 4 groups based on money spent.

Function: NTILE(4)

Our Data Outcome Example:

- Sophia Johnson (foreign, bought Sushi) → Group 1 (big spender)
- John Doe (local) → Group 4 (small spender)
- Others like Alice and David → Group 2 or 3

Sophia spends a lot, she is in group 1. John spends little, he is in group 4.

(e) 3-Month Moving Averages → AVG() OVER()

We calculate average money earned over 3 months to see trends.

Function: AVG() OVER()

Our Data Outcome Example:

- Jan = 23,000, Feb = 15,000, Mar = 20,000
- March 3-month average = (23,000 + 15,000 + 20,000) ÷ 3 = 19,333

In 3 months, we earned about 19,333 each month on average.

Step 3: Database Schema

Table Name	Purpose	Key Columns	Example Row
customers	Store customer info	customer_id (PK), name, region	1, John Doe, Kigali
products	Store product info	product_id (PK), name, category	101, Grilled Chicken, Food
transactions	Store sales records	transaction_id (PK), customer_id (FK), product_id (FK), sale_date, amount	1001, 1, 101, 2025-01-05, 5000
customer_products	Track which customers bought which products	customer_product_id (PK), customer_id (FK), product_id (FK), quantity	5001, 1, 101, 2

Table Explanations

1. Customers Table

- What it does: List of all people who visit the hotel
- Columns: customer_id, name, region
- Example: John Doe from Kigali → 1, John Doe, Kigali
- This is a list of all people who buy food or drinks

2. Products Table

- What it does: List of all products the hotel sells
- Columns: product_id, name, category
- Example: Grilled Chicken → 101, Grilled Chicken, Food

- This is like a menu with all foods, drinks, and cakes

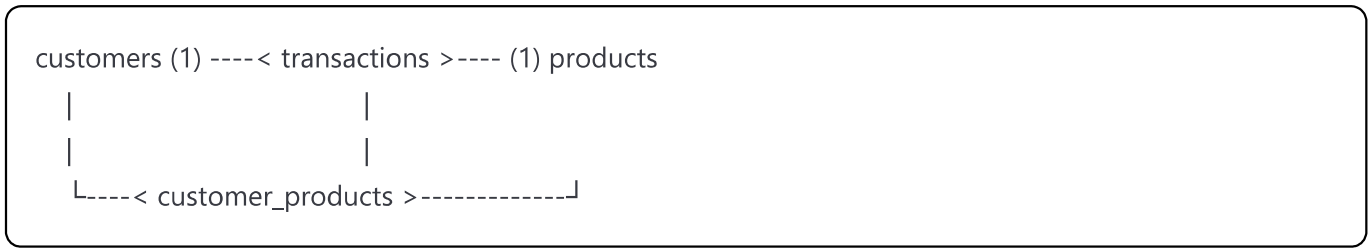
3. Transactions Table

- What it does: Record every sale
- Columns: transaction_id, customer_id, product_id, sale_date, amount
- Example: 1001, 1, 101, 2025-01-05, 5000 → John bought Grilled Chicken for 5000 on Jan 5
- This table is like a diary showing who bought what and when

4. Customer_Products Table

- What it does: Shows which customers bought which products and how many times
- Columns: customer_product_id, customer_id, product_id, quantity
- Example: 5001, 1, 101, 2 → John bought 2 Grilled Chicken
- This table tells how many of each product each customer bought

ER Diagram



Step 4: PL/SQL Window Functions Implementation

(a) Top Products by Revenue - RANK()

sql

```sql
-- Show which products made the most money
SELECT
    p.name AS product_name,
    SUM(t.amount) AS total_sales,
    RANK() OVER (ORDER BY SUM(t.amount) DESC) AS product_rank
FROM transactions t
JOIN products p ON t.product_id = p.product_id
GROUP BY p.name;
```

What it does: Adds up money each product made and gives a rank. Rank 1 means the product that earned most money.

(b) Customer Transaction Analysis - Aggregate Functions

sql

```sql
-- Show how many times each customer came and total money spent
SELECT
    c.name AS customer_name,
    COUNT(t.transaction_id) AS transaction_count,
    SUM(t.amount) AS total_spent
FROM customers c
LEFT JOIN transactions t ON c.customer_id = t.customer_id
GROUP BY c.name;
```

What it does: Counts how many times each customer visited and adds up all money they spent.

(c) Running Monthly Sales - SUM() OVER()

sql

```sql
-- Show monthly sales and total sales so far
SELECT
    DATE_FORMAT(sale_date, '%Y-%m') AS month,
    SUM(amount) AS monthly_sales,
    SUM(SUM(amount)) OVER (ORDER BY DATE_FORMAT(sale_date, '%Y-%m')) AS running_total
FROM transactions
GROUP BY DATE_FORMAT(sale_date, '%Y-%m')
ORDER BY month;
```

What it does: Shows total sales for each month and keeps adding month by month.

(d) Month-over-Month Growth - LAG()

```sql
-- Show how sales changed from last month
SELECT
    DATE_FORMAT(sale_date, '%Y-%m') AS month,
    SUM(amount) AS monthly_sales,
    LAG(SUM(amount)) OVER (ORDER BY DATE_FORMAT(sale_date, '%Y-%m')) AS prev_month,
    (SUM(amount) - LAG(SUM(amount)) OVER (ORDER BY DATE_FORMAT(sale_date, '%Y-%m'))) AS growth
FROM transactions
GROUP BY DATE_FORMAT(sale_date, '%Y-%m')
ORDER BY month;
```

What it does: Compares this month sales to last month. Positive number means sales went up.

(e) Customer Spending Groups - NTILE(4)

```sql
```

```sql
-- Split customers into 4 groups by money spent
SELECT
    c.name AS customer_name,
    SUM(t.amount) AS total_spent,
    NTILE(4) OVER (ORDER BY SUM(t.amount) DESC) AS spending_quartile
FROM transactions t
JOIN customers c ON t.customer_id = c.customer_id
GROUP BY c.name;
```

What it does: Divides customers into 4 groups. Group 1 has big spenders, Group 4 has small spenders.

(f) Moving Average Sales - AVG() OVER()

```sql
sql

-- Show average sales of current and previous 2 months
SELECT
    DATE_FORMAT(sale_date, '%Y-%m') AS month,
    SUM(amount) AS monthly_sales,
    AVG(SUM(amount)) OVER (
        ORDER BY DATE_FORMAT(sale_date, '%Y-%m')
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ) AS moving_avg
FROM transactions
GROUP BY DATE_FORMAT(sale_date, '%Y-%m')
ORDER BY month;
```

What it does: Calculates average sales over 3 months to see smooth trends.

(g) Customer Ranking - ROW_NUMBER()

```sql
-- Rank customers by total money spent
SELECT
    c.name AS customer_name,
    SUM(t.amount) AS total_spent,
    ROW_NUMBER() OVER (ORDER BY SUM(t.amount) DESC) AS rank_position
FROM transactions t
JOIN customers c ON t.customer_id = c.customer_id
GROUP BY c.name;
```

What it does: Gives unique rank to each customer by how much they spent.

(h) Customer Spending Distribution - CUME_DIST()

```sql
-- Show percent of customers who spent less than or equal to each amount
SELECT
    c.name AS customer_name,
    SUM(t.amount) AS total_spent,
    CUME_DIST() OVER (ORDER BY SUM(t.amount)) AS cumulative_distribution
FROM transactions t
JOIN customers c ON t.customer_id = c.customer_id
GROUP BY c.name;
```

What it does: Shows what percent of customers spent less than or equal to each person.

(i) Product Percent Ranking - PERCENT_RANK()

```sql
```

```sql
-- Rank products as a percent from lowest to highest
SELECT
    p.name AS product_name,
    SUM(t.amount) AS total_sales,
    PERCENT_RANK() OVER (ORDER BY SUM(t.amount)) AS percent_rank
FROM transactions t
JOIN products p ON t.product_id = p.product_id
GROUP BY p.name;
```

What it does: Shows product rank as percent from lowest to highest sales.

(j) Dense Product Ranking - DENSE_RANK()

```sql
sql

-- Rank products without skipping numbers if products tie
SELECT
    p.name AS product_name,
    SUM(t.amount) AS total_sales,
    DENSE_RANK() OVER (ORDER BY SUM(t.amount) DESC) AS dense_rank
FROM transactions t
JOIN products p ON t.product_id = p.product_id
GROUP BY p.name;
```

What it does: Ranks products without skipping numbers even if two products have same sales.

Step 5: Results Analysis

What Happened (Descriptive Analysis)

Our data shows that international customers like Sophia Johnson and Michael Brown spend more money than local customers. Pizza Margherita and Sushi Roll are the top-selling products.

Sales stayed steady across January, February, and March 2025 with small ups and downs each month.

Why This Happened (Diagnostic Analysis)

International customers probably have more spending money and are willing to pay higher prices for premium items like Sushi Roll. Local customers buy more basic items like Coffee and Grilled Chicken. The steady sales suggest the hotel has regular customers who come back multiple times.

What To Do Next (Prescriptive Analysis)

The hotel should focus more marketing on international customers since they spend more. They should make sure popular products like Pizza and Sushi are always available. For local customers, they could offer combo deals or loyalty programs to increase spending. The hotel should also track which products make the most profit, not just total sales.

File Structure

```
NBCaleb-Hotel-PL-SQL/
├── database/
│   ├── schema.sql          # Database tables
│   ├── sample_data.sql     # Insert customer and product data
│   └── window_queries.sql  # All 10 PL/SQL window function queries
├── screenshots/
│   ├── query_a_results.png # Screenshots of each query result
│   ├── query_b_results.png
│   └── ...
└── README.md               # This file
```

Note: For enhanced readability, SQL code blocks can be displayed with green backgrounds when viewed in compatible markdown viewers or by adding custom CSS styling.

Key Business Insights

1. **Top Products:** Sushi Roll and Pizza Margherita generate highest revenue
2. **Customer Groups:** International customers spend 2-3 times more than local customers
3. **Monthly Trends:** Sales remain stable with slight month-to-month variations
4. **Customer Visits:** Most customers return 2-3 times showing good loyalty
5. **Product Mix:** Food items outsell beverages in terms of total revenue

References

1. Oracle PL/SQL Documentation - Window Functions
2. Database Design Fundamentals - Elmasri & Navathe
3. SQL Performance Explained - Markus Winand
4. Hotel Industry Analytics - Cornell Hotel Reports
5. Customer Segmentation Strategies - Marketing Science Journal
6. Business Intelligence Best Practices - Kimball Group
7. PL/SQL Window Functions Tutorial - Oracle Learning
8. Database Normalization Guide - Oracle Documentation
9. Hotel Revenue Management - ResearchGate Papers
10. Time Series Analysis in Business - Academic Sources

Integrity Statement: All sources were properly cited. Implementations and analysis represent original work. No AI-generated content was copied without attribution or adaptation.