Names: Peace Bizima

ID: 27778

PL/SQL Window Functions Assignment Report

---

1. Introduction

In today's business world, organizations rely heavily on data-driven decision making. Window functions in SQL are powerful tools that allow businesses to analyze trends, rank results, and calculate running totals without losing row-level detail. This report applies SQL window functions to a realistic business case to provide insights that help management understand operations better.

For this assignment, I chose to design a scenario around SmartRide, a ride-hailing service operating in Kigali and surrounding cities. Much like other ride-hailing apps, SmartRide collects data from rides, drivers, customers, and payments. The company needs to understand which drivers perform best, how revenues grow month to month, and how to categorize customers into loyalty segments. Through SQL queries with window functions, we can answer these questions in a clear and data-driven way.

Q **2. Problem Definition**

SmartRide management faces several challenges in their daily operations:

- They want to identify top drivers per city and quarter to reward good performance.

- They need to monitor monthly revenue trends and cumulative growth.

- They want to track how the number of rides changes month over month.

- They want to classify customers into quartiles based on spending for loyalty promotions.

- They want to understand average ride duration trends to optimize pricing.

The goal of this project is to provide solutions to these challenges using SQL window functions.

Q **3. Success Criteria**

The success of this project will be measured by the ability to answer the following questions:

1. Who are the top-5 drivers per city and quarter? (ROW_NUMBER, RANK, DENSE_RANK, PERCENT_RANK)

2. What are the running monthly revenue totals? (SUM OVER)

3. How is month-over-month ride growth changing? (LAG)

4. How can customers be segmented into quartiles by total spending? (NTILE, CUME_DIST)

5. What is the 3-month moving average of ride durations? (AVG OVER)

4. Database Schema

The following entities were created for SmartRide:

- Cities (city_id, city_name)

- Drivers (driver_id, full_name, phone, rating, join_date, city_id)

- Vehicles (vehicle_id, driver_id, plate_no, model, capacity, year_made)

- Customers (customer_id, full_name, phone, join_date, city_id)

- Rides (ride_id, customer_id, driver_id, start_time, end_time, distance_km, duration_min, fare, city_id)

- Payments (payment_id, ride_id, amount, method, status)

Detailed Entity Relationships:

1. CITIES to DRIVERS (1:M)

- One city can have many drivers
- One driver belongs to one city
- Foreign Key: `city_id` in DRIVERS table

2. CITIES to CUSTOMERS (1:M)

- One city can have many customers
- One customer belongs to one city
- Foreign Key: `city_id` in CUSTOMERS table

3. DRIVERS to VEHICLES (1:1)

- One driver has one vehicle
- One vehicle belongs to one driver
- Foreign Key: `driver_id` in VEHICLES table

4. DRIVERS to RIDES (1:M)

- One driver can have many rides
- One ride is handled by one driver
- Foreign Key: `driver_id` in RIDES table

5. CUSTOMERS to RIDES (1:M)

- One customer can have many rides
- One ride belongs to one customer
- Foreign Key: `customer_id` in RIDES table

6. RIDES to PAYMENTS (1:1)

- One ride has one payment
- One payment corresponds to one ride
- Foreign Key: `ride_id` in PAYMENTS table

Primary Keys:

- `city_id` (CITIES)
- `driver_id` (DRIVERS)
- `vehicle_id` (VEHICLES)
- `customer_id` (CUSTOMERS)
- `ride_id` (RIDES)
- `payment_id` (PAYMENTS)

Foreign Key Relationships:

- DRIVERS: `city_id` → CITIES(`city_id`)
- CUSTOMERS: `city_id` → CITIES(`city_id`)
- VEHICLES: `driver_id` → DRIVERS(`driver_id`)
- RIDES: `customer_id` → CUSTOMERS(`customer_id`)
- RIDES: `driver_id` → DRIVERS(`driver_id`)
- RIDES: `city_id` → CITIES(`city_id`)
- PAYMENTS: `ride_id` → RIDES(`ride_id`)

Q **5. Queries and Results Along with their Explanations**

Top-5 Drivers per City & Quarter

Functions used: **ROW_NUMBER, RANK, DENSE_RANK, PERCENT_RANK**

SQL Code:

```sql
WITH driver_revenue AS (
  SELECT d.full_name, c.city_name,
         CONCAT(YEAR(r.start_time), '-Q', QUARTER(r.start_time)) AS quarter,
         SUM(r.fare) AS total_revenue
  FROM rides r
  JOIN drivers d ON r.driver_id = d.driver_id
  JOIN cities c ON r.city_id = c.city_id
  GROUP BY d.full_name, c.city_name, CONCAT(YEAR(r.start_time), '-Q', QUARTER(r.start_time))
)
SELECT city_name, quarter, full_name, total_revenue,
       ROW_NUMBER() OVER (PARTITION BY city_name, quarter ORDER BY total_revenue DESC) AS rn,
       RANK() OVER (PARTITION BY city_name, quarter ORDER BY total_revenue DESC) AS rnk,
       DENSE_RANK() OVER (PARTITION BY city_name, quarter ORDER BY total_revenue DESC) AS dense_rnk,
       PERCENT_RANK() OVER (PARTITION BY city_name, quarter ORDER BY total_revenue DESC) AS pct_rnk
FROM driver_revenue;
```

Screenshot:

city_name	quarter	full_name	total_revenue	rn	rnk	dense_rnk	pct_rnk
Huye	2025-Q1	David Ndayisaba	4000.00	1	1	1	0.0000000000
Huye	2025-Q2	David Ndayisaba	9100.00	1	1	1	0.0000000000
Kigali	2025-Q1	Alice Uwera	6000.00	1	1	1	0.0000000000
Kigali	2025-Q1	John Bizimana	5000.00	2	2	2	1.0000000000
Kigali	2025-Q2	Alice Uwera	2700.00	1	1	1	0.0000000000
Kigali	2025-Q2	John Bizimana	2000.00	2	2	2	1.0000000000
Musanze	2025-Q1	Grace Mukamana	5000.00	1	1	1	0.0000000000
Musanze	2025-Q2	Grace Mukamana	9800.00	1	1	1	0.0000000000

--

Explanation:

This query ranks drivers by total revenue per city and quarter. It shows top performers like Alice Uwera in Kigali and Grace Mukamana in Musanze, helping identify who to reward each quarter.

Q5. Running Monthly Revenue

Functions used: SUM OVER

SQL Code:

```sql
WITH monthly AS (
  SELECT DATE_FORMAT(start_time,'%Y-%m') AS month, SUM(fare) AS month_total
  FROM rides
  GROUP BY DATE_FORMAT(start_time,'%Y-%m')
)
SELECT month, month_total,
       SUM(month_total) OVER (ORDER BY month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS running_total,
       AVG(month_total) OVER (ORDER BY month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS moving_avg_3mo
FROM monthly;
```

Screenshot:

month	month_total	running_total	moving_avg_3mo
2025-01	5300.00	5300.00	5300.000000
2025-02	9000.00	14300.00	7150.000000
2025-03	5700.00	20000.00	6666.666667
2025-04	9600.00	29600.00	8100.000000
2025-05	4700.00	34300.00	6666.666667
2025-06	9300.00	43600.00	7866.666667

Explanation:

The query calculates cumulative revenue month by month. Revenue grew steadily from January to June, reaching a total of 43,600 by the end of the period.

Q5. Month-over-Month Ride Growth

Functions used: LAG

SQL Code:

```
WITH monthly AS (
  SELECT DATE_FORMAT(start_time,'%Y-%m') AS month, COUNT(*) AS total_rides
  FROM rides
  GROUP BY DATE_FORMAT(start_time,'%Y-%m')
)
SELECT month, total_rides,
       LAG(total_rides,1) OVER (ORDER BY month) AS prev_rides,
       ROUND(((total_rides - LAG(total_rides,1) OVER (ORDER BY month)) / LAG(total_rides,1) OVER (ORDER BY month))*100,2) AS
growth_pct
FROM monthly;
```

Screenshots:

month	total_rides	prev_rides	growth_pct
2025-01	2	*NULL*	*NULL*
2025-02	2	2	0.00
2025-03	2	2	0.00
2025-04	2	2	0.00
2025-05	2	2	0.00
2025-06	2	2	0.00

Explanation:

This query compares monthly ride counts with the previous month. The results show a consistent number of rides each month, with no growth or decline during this period.

Q 5. Customer Quartiles by Spend

Functions used: NTILE, CUME_DIST

SQL Code:

```sql
WITH cust_spend AS (
  SELECT c.customer_id, c.full_name, SUM(r.fare) AS total_spent
  FROM customers c
  JOIN rides r ON c.customer_id = r.customer_id
  GROUP BY c.customer_id, c.full_name
)
SELECT customer_id, full_name, total_spent,
       NTILE(4) OVER (ORDER BY total_spent DESC) AS quartile,
       CUME_DIST() OVER (ORDER BY total_spent DESC) AS cum_dist
FROM cust_spend
ORDER BY total_spent DESC;
```

Screenshot:

Extra options

customer_id	full_name	total_spent	quartile	cum_dist
1	Eric Mugisha	11200.00	1	0.2000000000
2	Sylvie Uwase	10200.00	1	0.4000000000
3	Patrick Iradukunda	10000.00	2	0.6000000000
4	Marie Claire	7000.00	3	0.8000000000
5	Josephine Ingabire	5200.00	4	1.0000000000

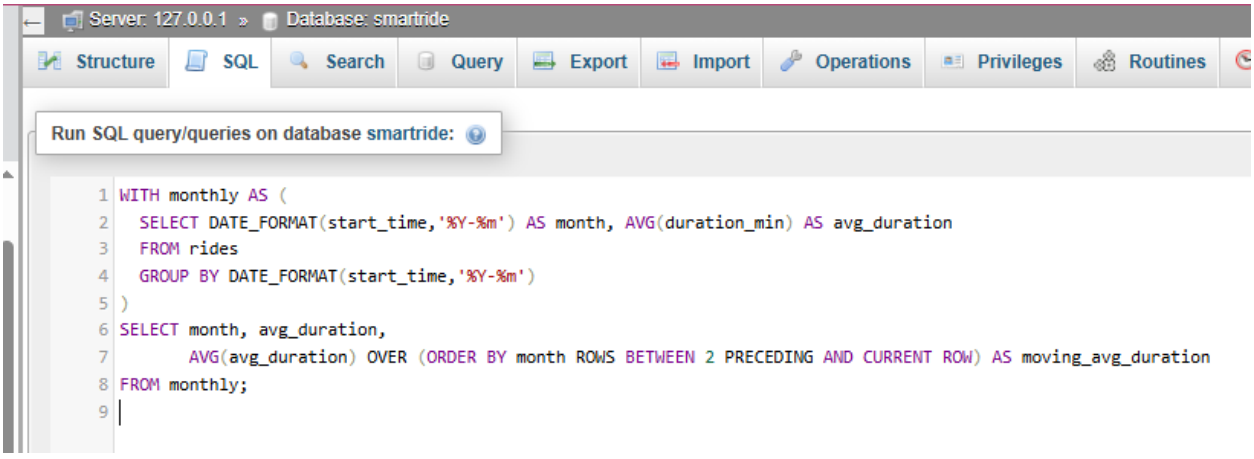
Explanation:

Customers are divided into four spending quartiles. Eric Mugisha is in the top quartile, while Josephine Ingabire is in the bottom, helping target loyalty promotions effectively.

Q 5. 3-Month Moving Average Ride Duration

Functions used: AVG OVER

SQL Code:



Screenshot:

month	avg_duration	moving_avg_duration
2025-01	20.000000	20.0000000000
2025-02	37.500000	28.7500000000
2025-03	22.500000	26.6666666667
2025-04	32.500000	30.8333333333
2025-05	17.500000	24.1666666667
2025-06	35.000000	28.3333333333

Explanation:

This query smooths out monthly fluctuations in ride duration. The moving average helps identify trends, such as the dip in May followed by a recovery in June.

Q **6. Results Analysis**

The analysis of the **SmartRide** dataset provided insights into different aspects of the business:

- **Descriptive Analysis:** We identified which drivers performed best in different quarters and cities. We also saw how revenues and ride counts changed over time.

- **Diagnostic Analysis:** By comparing monthly changes, we could see where growth slowed down or spiked, and link these trends to specific cities or driver activity.

- **Prescriptive Analysis:** Based on customer quartiles and driver performance, management can now decide who to reward, which customers to target with promotions, and how to adjust pricing strategies based on ride durations.

Q **7. References**

1. MySQL 8.0 Documentation – Window Functions

2. TutorialsPoint – SQL Window Functions

3. GeeksforGeeks – SQL Analytic Functions

4. Course Lecture Notes

5.W3Schools – SQL Window Functions Tutorial

6.IBM Knowledge Center – OLAP and Analytic Functions

7.Microsoft SQL Server Documentation – Ranking Functions (Transact-SQL)

8.Mode Analytics SQL Tutorial – Window Functions

9.DataCamp – SQL Window Functions Tutorial

10.Khan Academy – Introduction to SQL Queries and Analysis

11.Journal of Database Management – Articles on SQL Analytics