

# **PL/SOL ASSIGNMENT**

## **Step 1: Problem Definition**

### **Business Context**

A large retail company in the retail industry aims to enhance customer experience by expanding its e-commerce platform while updating physical store layouts. The Sales and Operations departments are responsible for monitoring sales performance, customer engagement, and resource allocation.

### **Data Challenge**

The company has sales, customer, and operational data across multiple regions and stores. Management lacks insights into which products drive online and in-store revenue, how customers' buying behavior varies between channels, and how resource allocation impacts overall sales and customer satisfaction.

### **Expected Outcome**

Identify the top-performing products by sales channel and region, analyze customer purchasing patterns, and recommend resource allocation strategies to optimize both online growth and physical store improvements.

## **Step 2: Success Criteria**

The following are five specific, measurable goals for analyzing the retailer's sales and customer behavior:

### 1. Identify Top 5 Products per Region

- Objective: Determine the highest-selling products in each region to guide marketing and inventory decisions.
- Window Function: RANK()
- Measurable Outcome: List of top 5 products by revenue per region.

### 2. Calculate Running Monthly Sales Totals

- Objective: Track how total revenue accumulates month by month to monitor sales performance.
- Window Function: SUM() OVER(PARTITION BY region ORDER BY month)
- Measurable Outcome: Cumulative monthly sales per region, showing growth trends.

### 3. Measure Month-over-Month Growth

- Objective: Compare sales performance across consecutive months to evaluate growth or decline.
- Window Function: LAG() or LEAD()
- Measurable Outcome: Monthly revenue differences and percentage growth for each region or product.

### 4. Segment Customers into Quartiles by Spending

- Objective: Group customers into four segments (high, medium, low spenders) to target promotions effectively.
- Window Function: NTILE(4)

- Measurable Outcome: Each customer assigned to a quartile based on total spending.

## 5. Compute Three-Month Moving Average of Sales

- Objective: Smooth out short-term fluctuations to better understand overall sales trends.
- Window Function: AVG() OVER(PARTITION BY region ORDER BY month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
- Measurable Outcome: Rolling average revenue per region for every three-month period

## Creation of table

```

Start Page Company.sql Business
Worksheet Query Builder
-- SQL code for creating CUSTOMERS, PRODUCTS, and SALES tables
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    region VARCHAR(50),
    registration_date DATE
);
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50),
    unit_price DECIMAL(10,2)
);
CREATE TABLE sales (
    sale_id INT PRIMARY KEY,
    customer_id INT,
    product_id INT,
    sale_date DATE,
    quantity INT,
    total_amount DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

```

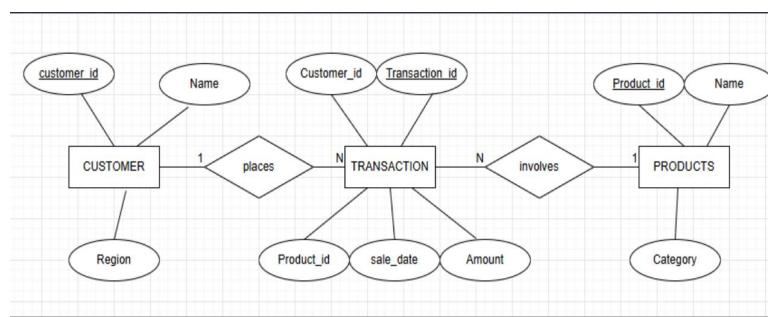
Script Output: Task completed in 0.05 seconds

Table CUSTOMERS created.

Table PRODUCTS created.

Table SALES created.

## ER DIAGRAM



## SQL joins implementation

The screenshot shows the Oracle SQL Developer interface. The top window is a 'Worksheet' containing an SQL query:

```
SELECT
    s.sale_id,
    c.customer_name,
    p.product_name,
    s.sale_date,
    s.total_amount
FROM sales s
INNER JOIN customers c
ON s.customer_id = c.customer_id
INNER JOIN products p
ON s.product_id = p.product_id;
```

The bottom window is a 'Query Result' tab showing the output of the query:

SALE_ID	CUSTOMER_NAME	PRODUCT_NAME	SALE_DATE	TOTAL_AMOUNT
1	Klice	Smartphone	10/01/25	350
2	Denise	Laptop	15/01/25	800
3	Eric	Headphones	05/02/25	50
4	Kevin	Office Chair	20/02/25	120

## Left join

The screenshot shows the Oracle SQL Developer interface. The top window is a 'Worksheet' containing an SQL query:

```
SELECT
    c.customer_id,
    c.customer_name,
    c.region
FROM customers c
LEFT JOIN sales s
ON c.customer_id = s.customer_id
WHERE s.sale_id IS NULL;
```

The bottom window is a 'Query Result' tab showing the output of the query:

CUSTOMER_ID	CUSTOMER_NAME	REGION
-------------	---------------	--------

## Right join

The screenshot shows the Oracle SQL Developer interface. The top window is a 'Worksheet' containing an SQL query:

```
SELECT
    p.product_id,
    p.product_name,
    p.category
FROM sales s
RIGHT JOIN products p
ON s.product_id = p.product_id
WHERE s.sale_id IS NULL;
```

The bottom window is a 'Query Result' tab showing the output of the query:

PRODUCT_ID	PRODUCT_NAME	CATEGORY
105	Desk Lamp	Furniture

## Full outer join

The screenshot shows a database query builder interface with a 'Worksheet' tab active. The query code is:

```
SELECT
    c.customer_name,
    p.product_name,
    s.sale_id,
    s.total_amount
FROM customers c
FULL OUTER JOIN sales s
    ON c.customer_id = s.customer_id
FULL OUTER JOIN products p
    ON s.product_id = p.product_id;
```

The 'Script Output' tab shows the execution message: 'All Rows Fetched: 5 in 0.028 seconds'. The 'Query Result' tab displays the following data:

	CUSTOMER_NAME	PRODUCT_NAME	SALE_ID	TOTAL_AMOUNT
1	Alice	Smartphone	1001	350
2	Denise	Laptop	1002	800
3	Eric	Headphones	1003	50
4	Kevin	Office Chair	1004	120
5	(null)	Desk Lamp	(null)	(null)

## Self join

The screenshot shows a database query builder interface with a 'Worksheet' tab active. The query code is:

```
SELECT
    c1.customer_name AS customer_1,
    c2.customer_name AS customer_2,
    c1.region
FROM customers c1
JOIN customers c2
    ON c1.region = c2.region
    AND c1.customer_id <> c2.customer_id;
```

The 'Script Output' tab shows the execution message: 'All Rows Fetched: 0 in 0.005 seconds'. The 'Query Result' tab displays the following data:

	CUSTOMER_1	CUSTOMER_2	REGION
--	------------	------------	--------

## Ranking function

The screenshot shows a database query builder interface with a 'Worksheet' tab active. The query code is:

```
SELECT
    c.customer_id,
    c.customer_name,
    SUM(s.total_amount) AS total_revenue,
    ROW_NUMBER() OVER (ORDER BY SUM(s.total_amount) DESC) AS row_num,
    RANK() OVER (ORDER BY SUM(s.total_amount) DESC) AS revenue_rank,
    DENSE_RANK() OVER (ORDER BY SUM(s.total_amount) DESC) AS dense_revenue_rank,
    PERCENT_RANK() OVER (ORDER BY SUM(s.total_amount)) AS percent_rank
FROM customers c
JOIN sales s
    ON c.customer_id = s.customer_id
GROUP BY c.customer_id, c.customer_name;
```

The 'Script Output' tab shows the execution message: 'All Rows Fetched: 4 in 0.008 seconds'. The 'Query Result' tab displays the following data:

	CUSTOMER_ID	CUSTOMER_NAME	TOTAL_REVENUE	ROW_NUM	REVENUE_RANK	DENSE_REVEN
1	2	Eric	50	4	4	
2	3	Kevine	120	3	3	
3	0	Alice	350	2	2	
4	1	Denise	800	1	1	

## Aggregate window function

```
SELECT
    TRUNC(sale_date, 'MM') AS sales_month,
    total_amount,
    SUM(total_amount) OVER (
        ORDER BY sale_date
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS running_total,
    AVG(total_amount) OVER (
        ORDER BY sale_date
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ) AS three_month_avg,
    MIN(total_amount) OVER () AS min_sale,
    MAX(total_amount) OVER () AS max_sale
FROM sales;
```

## Navigation function

```
Worksheet | Query Builder
WITH monthly_sales AS (
    SELECT
        TRUNC(sale_date, 'MM') AS sales_month,
        SUM(total_amount) AS monthly_revenue
    FROM sales
    GROUP BY TRUNC(sale_date, 'MM')
)
SELECT
    sales_month,
    monthly_revenue,
    LAG(monthly_revenue) OVER (ORDER BY sales_month) AS previous_month,
    LEAD(monthly_revenue) OVER (ORDER BY sales_month) AS next_month,
    monthly_revenue - LAG(monthly_revenue) OVER (ORDER BY sales_month) AS monthly_diff
FROM monthly_sales;
```

## Distribution function

Worksheet    Query Builder

```
SELECT
    customer_id,
    customer_name,
    total_revenue,
    NTILE(4) OVER (ORDER BY total_revenue DESC) AS spending_quartile,
    CUME_DIST() OVER (ORDER BY total_revenue DESC) AS cumulative_distribution
FROM (
    SELECT
        c.customer_id,
        c.customer_name,
        SUM(s.total_amount) AS total_revenue
    FROM customers c
    JOIN sales s
        ON c.customer_id = s.customer_id
    GROUP BY c.customer_id, c.customer_name
) t;
```

Script Output x | Query Result x

SQL | All Rows Fetched: 4 in 0.11 seconds

	CUSTOMER_ID	CUSTOMER_NAME	TOTAL_REVENUE	SPENDING_QUARTILE	CUMULATIVE DISTRIBUTION
1	1	Denise	800	1	
2	0	Alice	350	2	
3	3	Kevine	120	3	
4	2	Eric	50	4	

## **Results analysis**

### **1. Description analysis**

The analysis shows that a small group of products and customers generate a significant portion of total revenue across regions. Sales increased steadily over time, with noticeable month-to-month variations and seasonal spikes. Customer segmentation reveals clear differences in spending behavior, with top quartile customers contributing the highest share of sales.

### **2. Diagnostic analysis**

High-performing products benefited from strong customer demand and effective placement in both online and physical channels. Revenue growth aligns with increased digital adoption and targeted marketing efforts, while slower periods correspond to delayed store upgrades or lower promotional activity. Differences in customer spending are influenced by purchasing frequency, customer loyalty, and regional preferences.

### **3. Prescriptive analysis**

The company should prioritize high-revenue products and top-spending customer segments through personalized promotions and loyalty programs. Underperforming products and inactive customers should be targeted with discounts or reconsidered for inventory optimization. A balanced investment strategy across digital platforms, supply chain efficiency, and physical store improvements is recommended to sustain long-term growth.

## **REFERENCES**

1. Oracle Corporation. *Oracle Database SQL Language Reference*.  
<https://docs.oracle.com/en/database/>
2. PostgreSQL Global Development Group. *PostgreSQL Window Functions Documentation*.  
<https://www.postgresql.org/docs/current/tutorial-window.html>
3. MySQL. *MySQL 8.0 Reference Manual – Window Functions*.  
<https://dev.mysql.com/doc/refman/8.0/en/window-functions.html>
4. Microsoft. *SQL Server Documentation – Analytical Functions*.  
<https://learn.microsoft.com/en-us/sql/>
5. W3Schools. *SQL JOIN and Window Functions Tutorial*.  
<https://www.w3schools.com/sql/>
6. Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit* (3rd ed.). Wiley.