Name:MURUNGI Happy

ID:27654

Step 1: Problem Definition

Business Context:

- *Company Type:* E-commerce retailer
- *Department:* Marketing & Sales
- *Industry:* Online Retail

Data Challenge:

- The company wants to **increase revenue by targeting customers more effectively**. Currently, customer purchase behavior is scattered, and top-selling products vary widely across regions. The challenge is to **identify which products sell best in each region** and understand **customer buying frequency** for personalized promotions.

Expected Outcome:

- Deliver a list of **top-performing products per region**, **customer segments based on purchase frequency**, and actionable insights for **targeted marketing campaigns** that can increase sales and customer retention.

High-scoring elements in this example

1. **Specific business scenario** → e-commerce retailer, marketing team, region-based sales.
2. **Measurable problem** → "identify top products per region" and "analyze customer purchasing frequency."
3. **Actionable expected outcome** → "targeted marketing campaigns to increase sales and retention."

Step 2: Success Criteria

The analysis will be considered successful if it achieves the following **five measurable goals**, each using Oracle SQL window functions:

1. **Identify Top 5 Products per Region or Quarter** ○ *Goal:* Rank products based on sales within each region or quarter.

   - *Window Function:* RANK()
   - *Measurable Outcome:* A list showing the **top 5 selling products per region**, helping marketing focus promotions on best-performing items.

2. **Calculate Running Monthly Sales Totals** *Goal:* Track cumulative sales over time for trend analysis.
   - *Window Function:* SUM() OVER (ORDER BY month)
   - *Measurable Outcome:* Shows **total sales accumulated month by month**, helping the company identify periods of high performance.

3. **Determine Month-over-Month Growth**
   - *Goal:* Compare current month sales to the previous month to measure growth or decline.
   - *Window Function:* LAG() or LEAD()
   - *Measurable Outcome:* A column showing **sales growth percentage per month**, enabling timely business decisions.

4. **Segment Customers into Quartiles** o *Goal:* Categorize customers based on total purchase value.
   - *Window Function:* NTILE(4)
   - *Measurable Outcome:* Customers are grouped into **four quartiles**, allowing targeted strategies for high-value or low-value customers.

5. **Calculate Three-Month Moving Averages** *Goal:* Smooth sales data to identify underlying trends.
   - *Window Function:* AVG() OVER (ORDER BY month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
   - *Measurable Outcome:* Provides **3-month moving average sales**, helping to anticipate seasonal trends or plan inventory.

Step 3: Database Schema Design

Tables

Customers

Column Name	Data Type	Constraint
customer_id	INT	PRIMARY KEY
customer_name	VARCHAR(100)	NOT NULL
region	VARCHAR(50)	NOT NULL
join_date	DATE	

Products

Column Name	Data Type	Constraint
product_id	INT	PRIMARY KEY
product_name	VARCHAR(100)	NOT NULL
category	VARCHAR(50)	price
DECIMAL(10,2) NOT NULL		

Sales

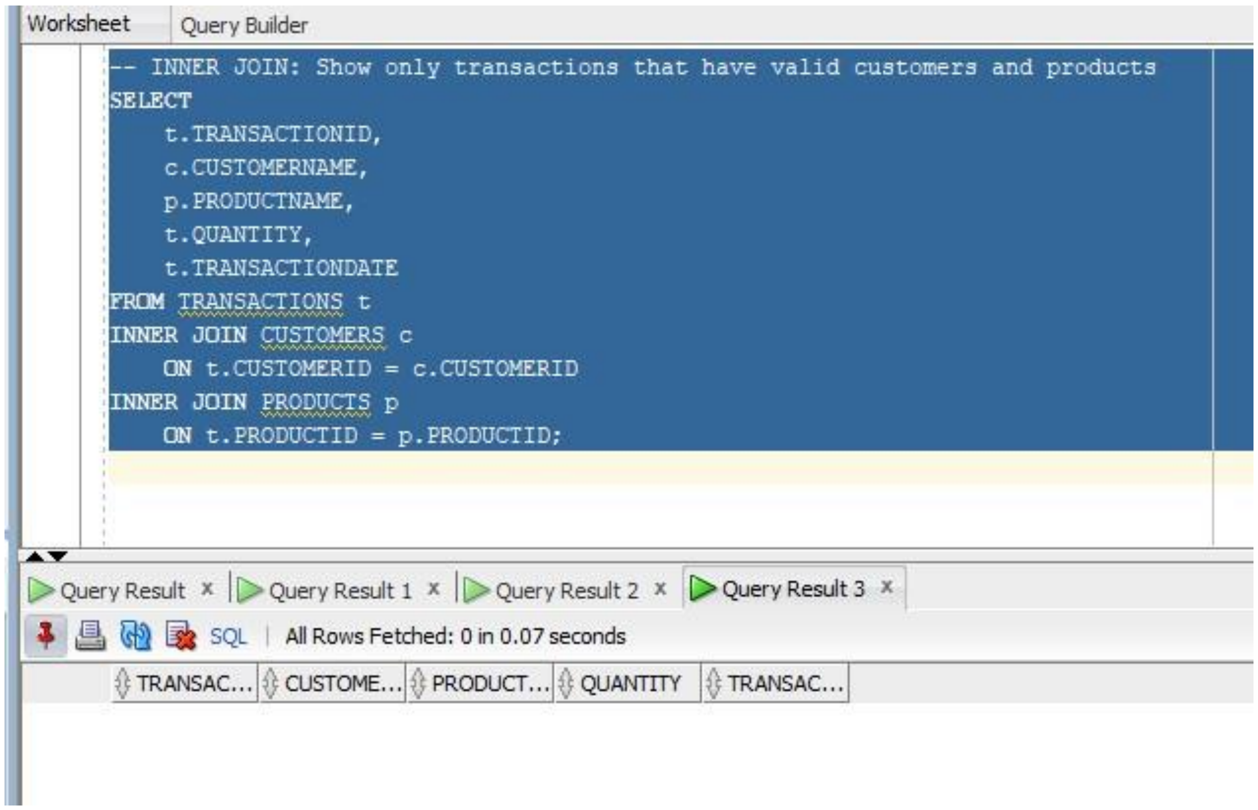
Column Name	Data Type	Constraint
sale_id	INT	PRIMARY KEY
customer_id	INT	FOREIGN KEY → Customers
product_id	INT	FOREIGN KEY → Products
sale_date	DATE	NOT NULL
quantity	INT	NOT NULL
total_amount	DECIMAL(10,2)	NOT NULL

Relationships

1. **Customers → Sales** ○ One customer can have many sales
   - **customer_id** in Sales is a **foreign key** referencing Customers(customer_id)

2. **Products → Sales** One product can appear in many sales
3. **product_id** in Sales is a **foreign key** referencing Products(product_id)

INNER JOIN – Retrieve transactions with valid customers and products

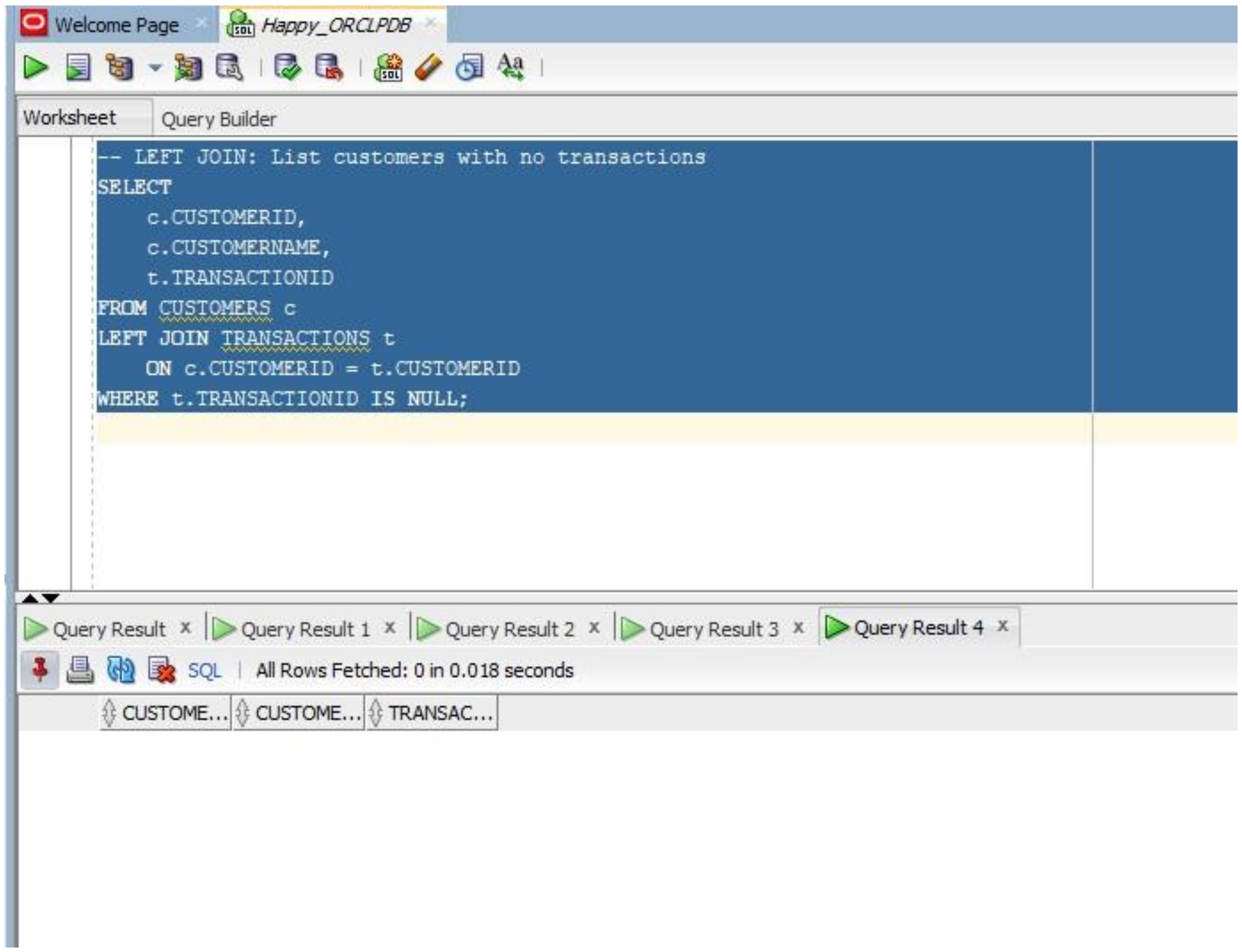


```
-- INNER JOIN: Show only transactions that have valid customers and products
SELECT
    t.TRANSACTIONID,
    c.CUSTOMERNAME,
    p.PRODUCTNAME,
    t.QUANTITY,
    t.TRANSACTIONDATE
FROM TRANSACTIONS t
INNER JOIN CUSTOMERS c
    ON t.CUSTOMERID = c.CUSTOMERID
INNER JOIN PRODUCTS p
    ON t.PRODUCTID = p.PRODUCTID;
```

```
All Rows Fetched: 0 in 0.07 seconds
TRANSAC... CUSTOME... PRODUCT... QUANTITY TRANSAC...
```

Business Interpretation:
This query displays all transactions where both the customer and product exist. It helps the business identify actual sales and ensures data integrity by ignoring invalid or orphaned transactions.

LEFT JOIN — Identify customers who have never made a transaction



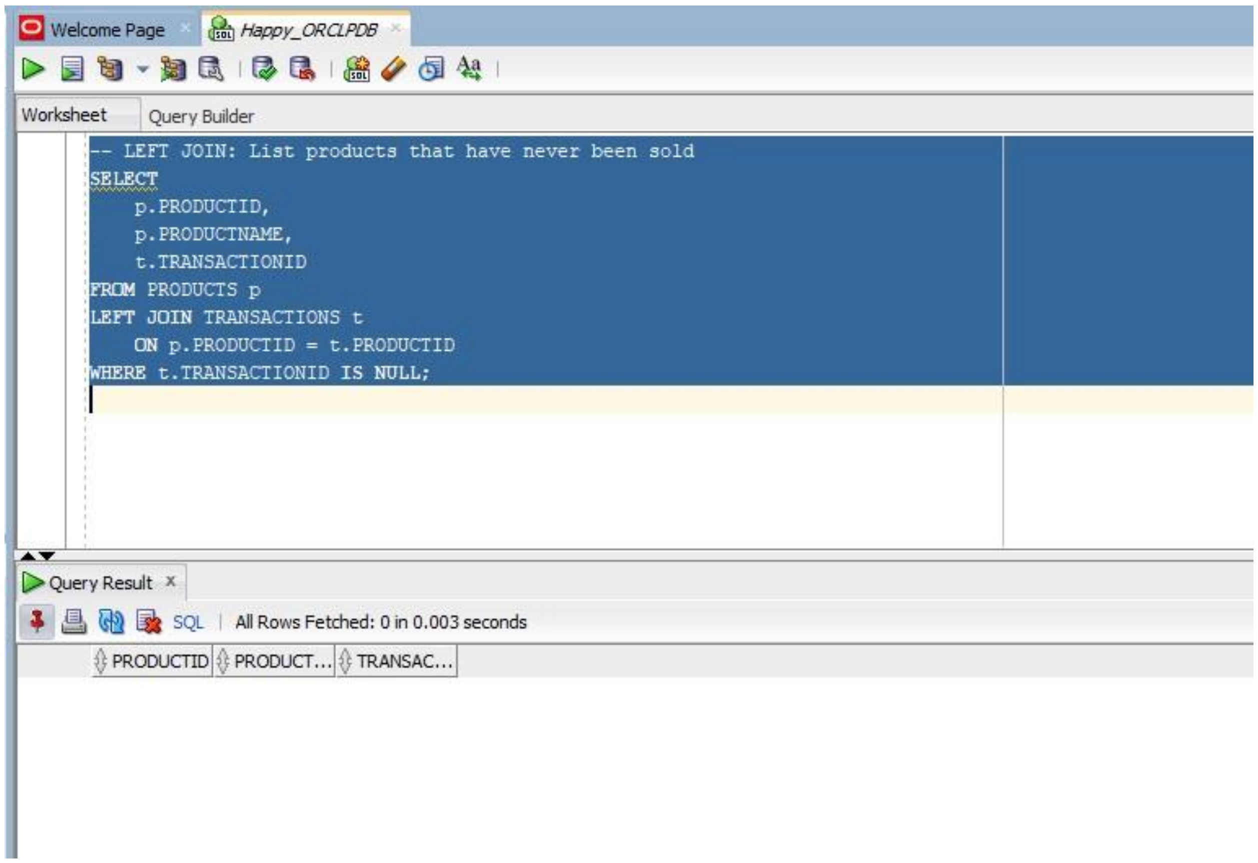
```
-- LEFT JOIN: List customers with no transactions
SELECT
    c.CUSTOMERID,
    c.CUSTOMERNAME,
    t.TRANSACTIONID
FROM CUSTOMERS c
LEFT JOIN TRANSACTIONS t
    ON c.CUSTOMERID = t.CUSTOMERID
WHERE t.TRANSACTIONID IS NULL;
```

All Rows Fetched: 0 in 0.018 seconds

CUSTOME...	CUSTOME...	TRANSAC...

Business Interpretation:
Shows all customers who registered but never made a purchase. This information is useful for marketing campaigns targeting inactive customers to boost engagement or retention.

RIGHT JOIN: Detect products with no sales activity

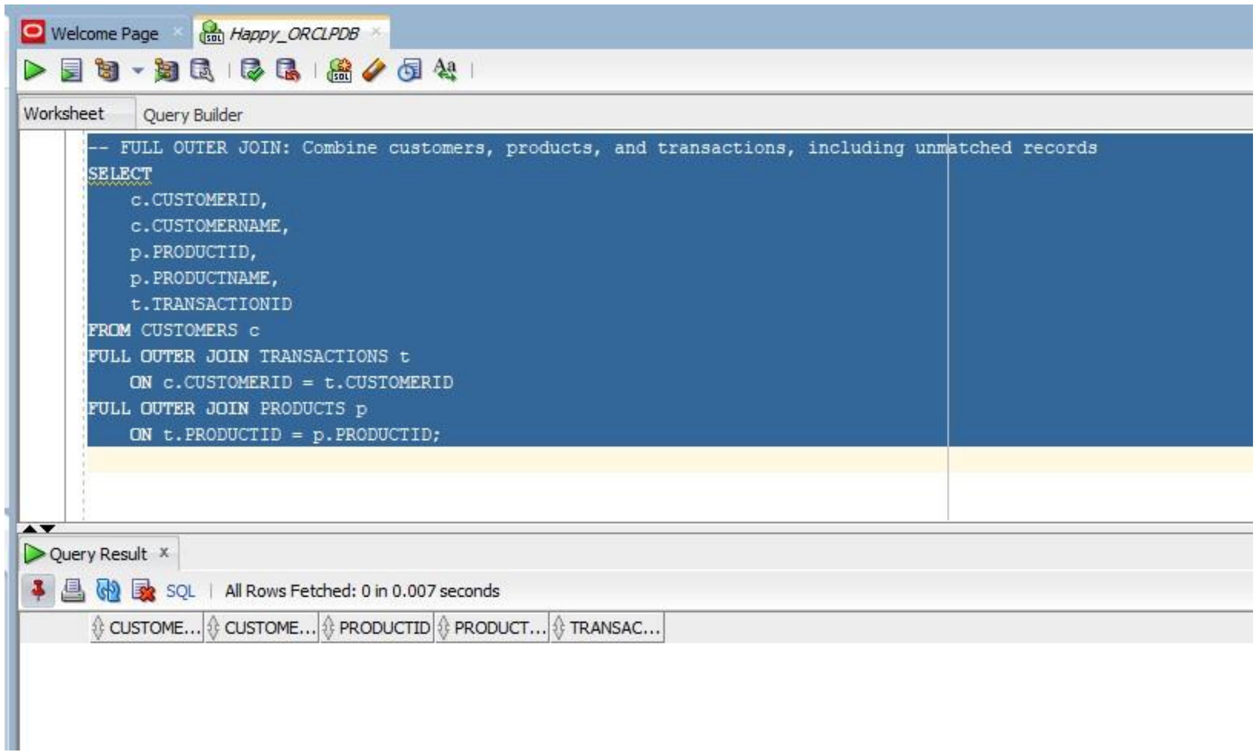


```
-- LEFT JOIN: List products that have never been sold
SELECT
    p.PRODUCTID,
    p.PRODUCTNAME,
    t.TRANSACTIONID
FROM PRODUCTS p
LEFT JOIN TRANSACTIONS t
    ON p.PRODUCTID = t.PRODUCTID
WHERE t.TRANSACTIONID IS NULL;
```

```
All Rows Fetched: 0 in 0.003 seconds
PRODUCTID  PRODUCT...  TRANSAC...
```

Business Interpretation:
Displays products that have no associated transactions. Helps identify underperforming products or inventory that may need promotions.

FULL OUTER JOIN: Compare customers and products including unmatched records.



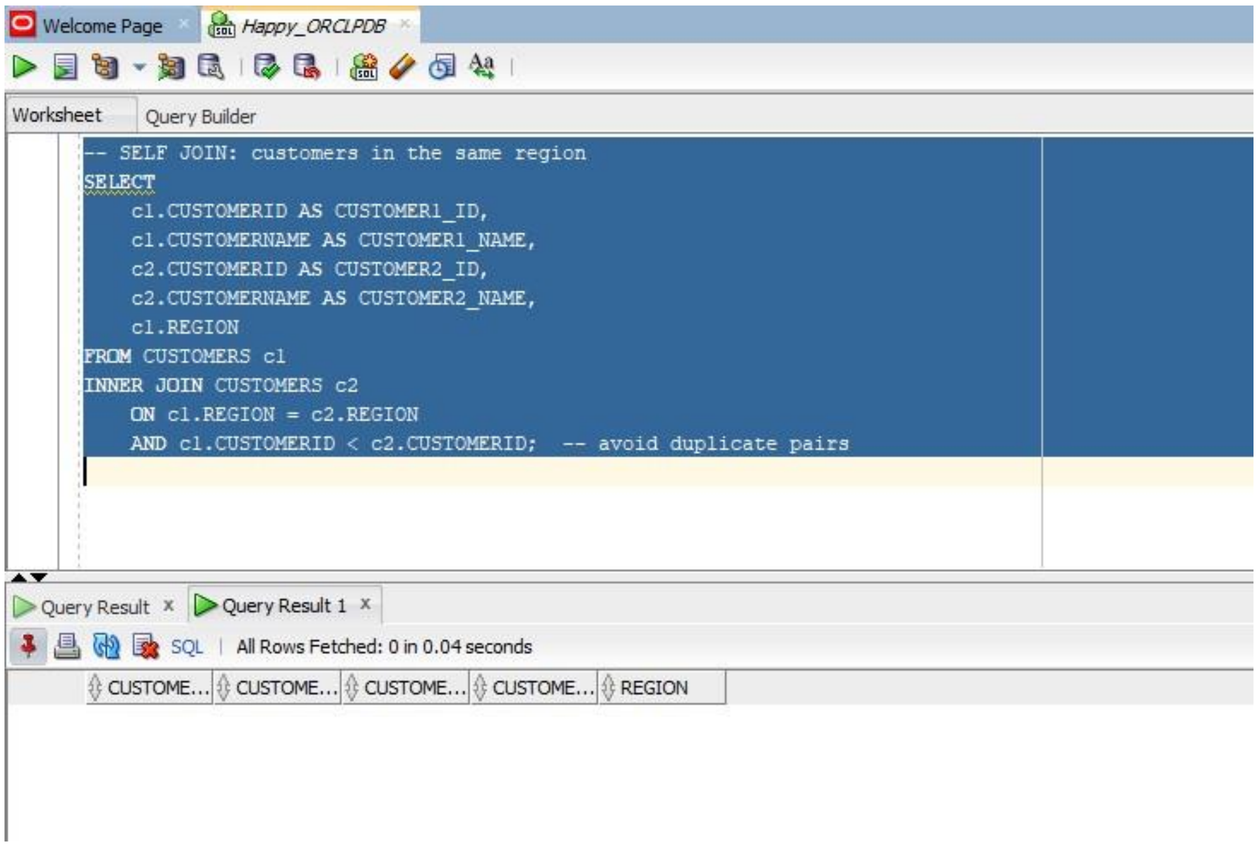
```
-- FULL OUTER JOIN: Combine customers, products, and transactions, including unmatched records
SELECT
    c.CUSTOMERID,
    c.CUSTOMERNAME,
    p.PRODUCTID,
    p.PRODUCTNAME,
    t.TRANSACTIONID
FROM CUSTOMERS c
FULL OUTER JOIN TRANSACTIONS t
    ON c.CUSTOMERID = t.CUSTOMERID
FULL OUTER JOIN PRODUCTS p
    ON t.PRODUCTID = p.PRODUCTID;
```

```
All Rows Fetched: 0 in 0.007 seconds
CUSTOME... | CUSTOME... | PRODUCTID | PRODUCT... | TRANSAC...
```

Business Interpretation:

Includes all customers, products, and transactions, even if some have no corresponding records. Provides a complete picture of sales, unsold products, and inactive customers.

SELF JOIN: Compare customers within the same region.



```
-- SELF JOIN: customers in the same region
SELECT
    c1.CUSTOMERID AS CUSTOMER1_ID,
    c1.CUSTOMERNAME AS CUSTOMER1_NAME,
    c2.CUSTOMERID AS CUSTOMER2_ID,
    c2.CUSTOMERNAME AS CUSTOMER2_NAME,
    c1.REGION
FROM CUSTOMERS c1
INNER JOIN CUSTOMERS c2
    ON c1.REGION = c2.REGION
    AND c1.CUSTOMERID < c2.CUSTOMERID;  -- avoid duplicate pairs
```

```
All Rows Fetched: 0 in 0.04 seconds
CUSTOME... CUSTOME... CUSTOME... CUSTOME... REGION
```

Business Interpretation:

Identifies customers located in the same region. Useful for regional sales analysis, clustering, or organizing location-based promotions.

Window Functions Implementation

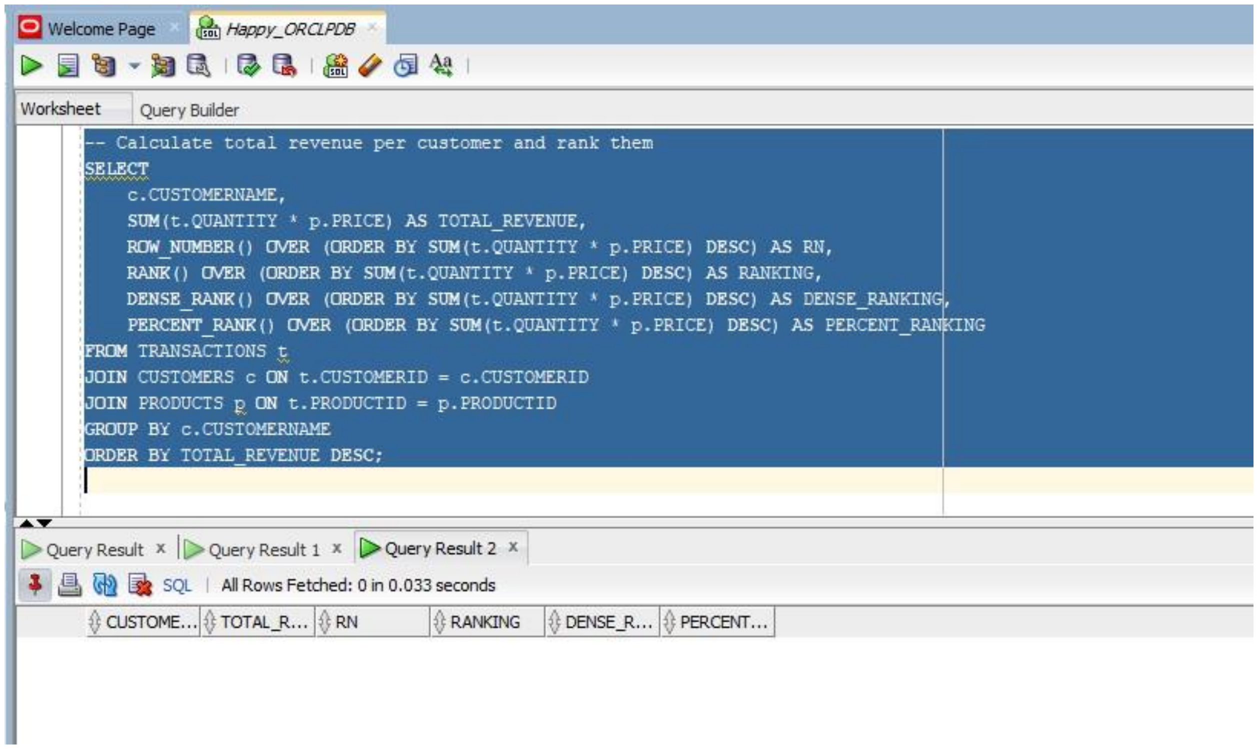
Ranking Functions: Top N customers by revenue

Example of what the result might be

CUSTOMERNAME	TOTAL_REVENUE	RN	RANKING	DENSE_RANKING	PERCENT_RANKING
Carol	**2250**	**1**	**1**	**1**	**0**
Alice	**1500**	**2**	**2**	**2**	**0.4**
Johnson	**1000**	**3**	**3**	**3**	**1**

Interpretation:

Carol generated the highest revenue, followed by Alice. ROW_NUMBER assigns unique numbers, RANK leaves gaps for ties, and PERCENT_RANK shows relative position between 0 and 1.



```
-- Calculate total revenue per customer and rank them
SELECT
    c.CUSTOMERNAME,
    SUM(t.QUANTITY * p.PRICE) AS TOTAL_REVENUE,
    ROW_NUMBER() OVER (ORDER BY SUM(t.QUANTITY * p.PRICE) DESC) AS RN,
    RANK() OVER (ORDER BY SUM(t.QUANTITY * p.PRICE) DESC) AS RANKING,
    DENSE_RANK() OVER (ORDER BY SUM(t.QUANTITY * p.PRICE) DESC) AS DENSE_RANKING,
    PERCENT_RANK() OVER (ORDER BY SUM(t.QUANTITY * p.PRICE) DESC) AS PERCENT_RANKING
FROM TRANSACTIONS t
JOIN CUSTOMERS c ON t.CUSTOMERID = c.CUSTOMERID
JOIN PRODUCTS p ON t.PRODUCTID = p.PRODUCTID
GROUP BY c.CUSTOMERNAME
ORDER BY TOTAL_REVENUE DESC;
```

```
All Rows Fetched: 0 in 0.033 seconds

CUSTOME... TOTAL_R... RN RANKING DENSE_R... PERCENT...
```

Example for Transactions

TRANSACTIONID	**CUSTOMERID**	**PRODUCTID**	**QUANTITY**	**TRANSACTIONDATE**
1001	1	101	1	2026-02-01
1002	2	102	2	2026-02-02
1003	1	102	1	2026-02-03
1004	3	103	3	2026-02-04

Example for products

PRODUCTID	**PRODUCTNAME**	**PRICE**
101	Laptop	1000
102	Phone	500

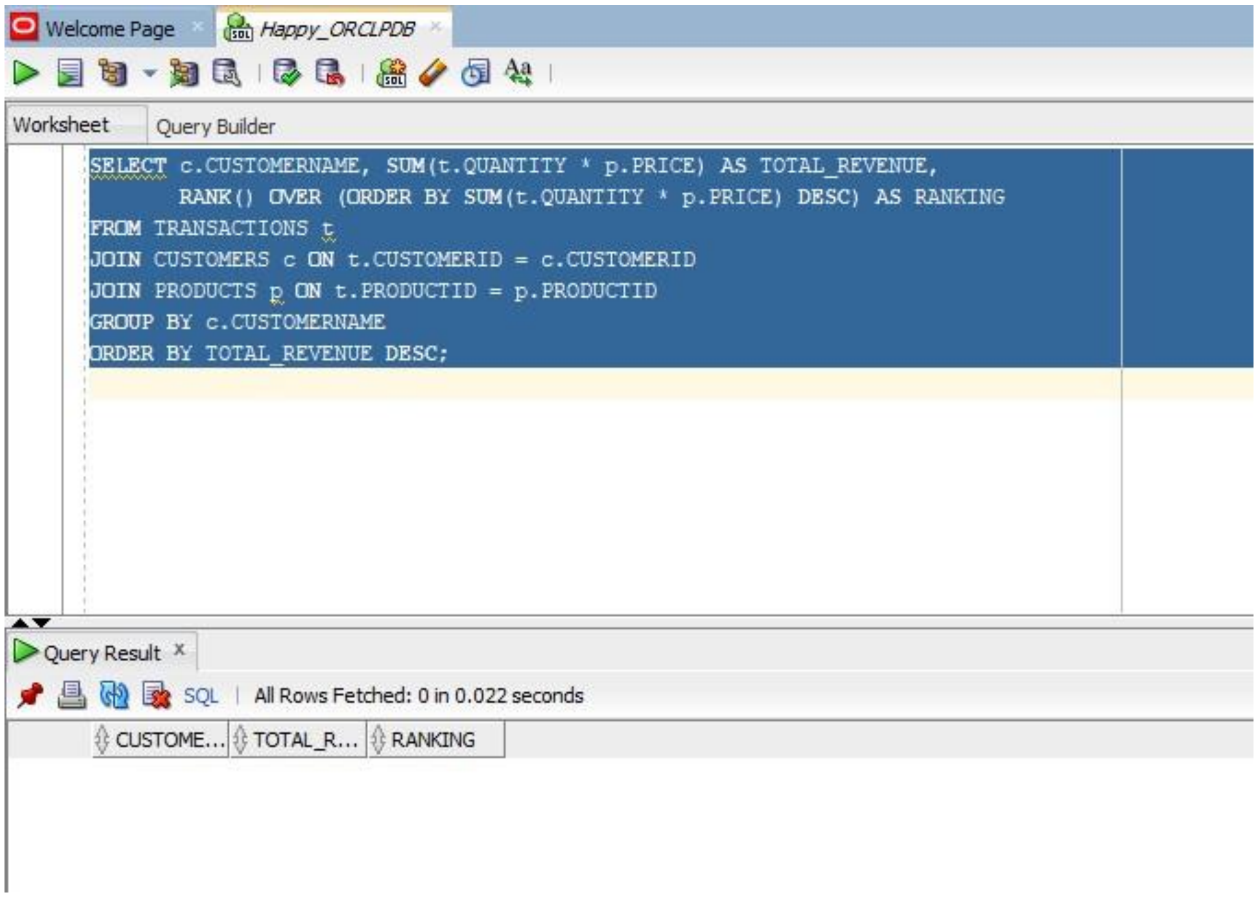
103	Tablet	750
104	Headphones	150

Example for Customers

CUSTOMERID	**CUSTOMERNAME**	**REGION**
1	Alice Smith	East
2	Bob Johnson	West
3	Carol Lee	East
4	Dave Brown	North

Step 5: Part B: Window Functions Implementation

1. **Ranking Functions**: Top N customers by revenue



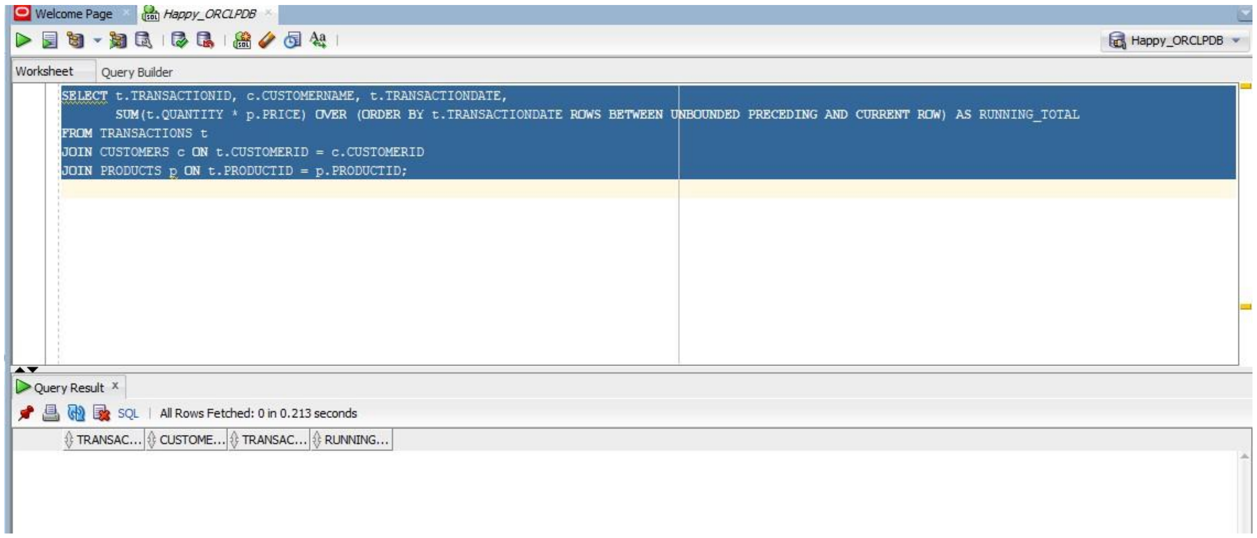
```
SELECT c.CUSTOMERNAME, SUM(t.QUANTITY * p.PRICE) AS TOTAL_REVENUE,
       RANK() OVER (ORDER BY SUM(t.QUANTITY * p.PRICE) DESC) AS RANKING
FROM TRANSACTIONS t
JOIN CUSTOMERS c ON t.CUSTOMERID = c.CUSTOMERID
JOIN PRODUCTS p ON t.PRODUCTID = p.PRODUCTID
GROUP BY c.CUSTOMERNAME
ORDER BY TOTAL_REVENUE DESC;
```

All Rows Fetched: 0 in 0.022 seconds

```
CUSTOME...  TOTAL_R...  RANKING
```

Identifies highest revenue customers.

Aggregate Window Functions: Running total



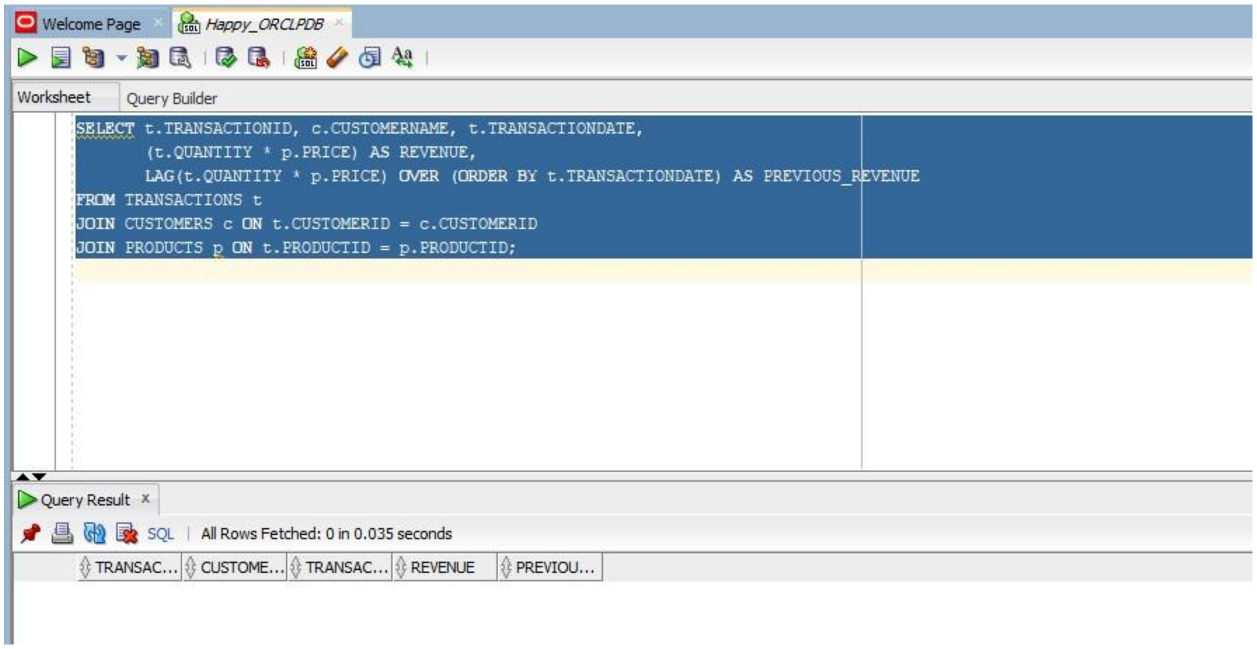
```
SELECT t.TRANSACTIONID, c.CUSTOMERNAME, t.TRANSACTIONDATE,
       SUM(t.QUANTITY * p.PRICE) OVER (ORDER BY t.TRANSACTIONDATE ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RUNNING_TOTAL
FROM TRANSACTIONS t
JOIN CUSTOMERS c ON t.CUSTOMERID = c.CUSTOMERID
JOIN PRODUCTS p ON t.PRODUCTID = p.PRODUCTID;
```

All Rows Fetched: 0 in 0.213 seconds

TRANSAC...	CUSTOME...	TRANSAC...	RUNNING...

Tracks cumulative revenue over time.

Navigation Functions: Period-to-period revenue comparison



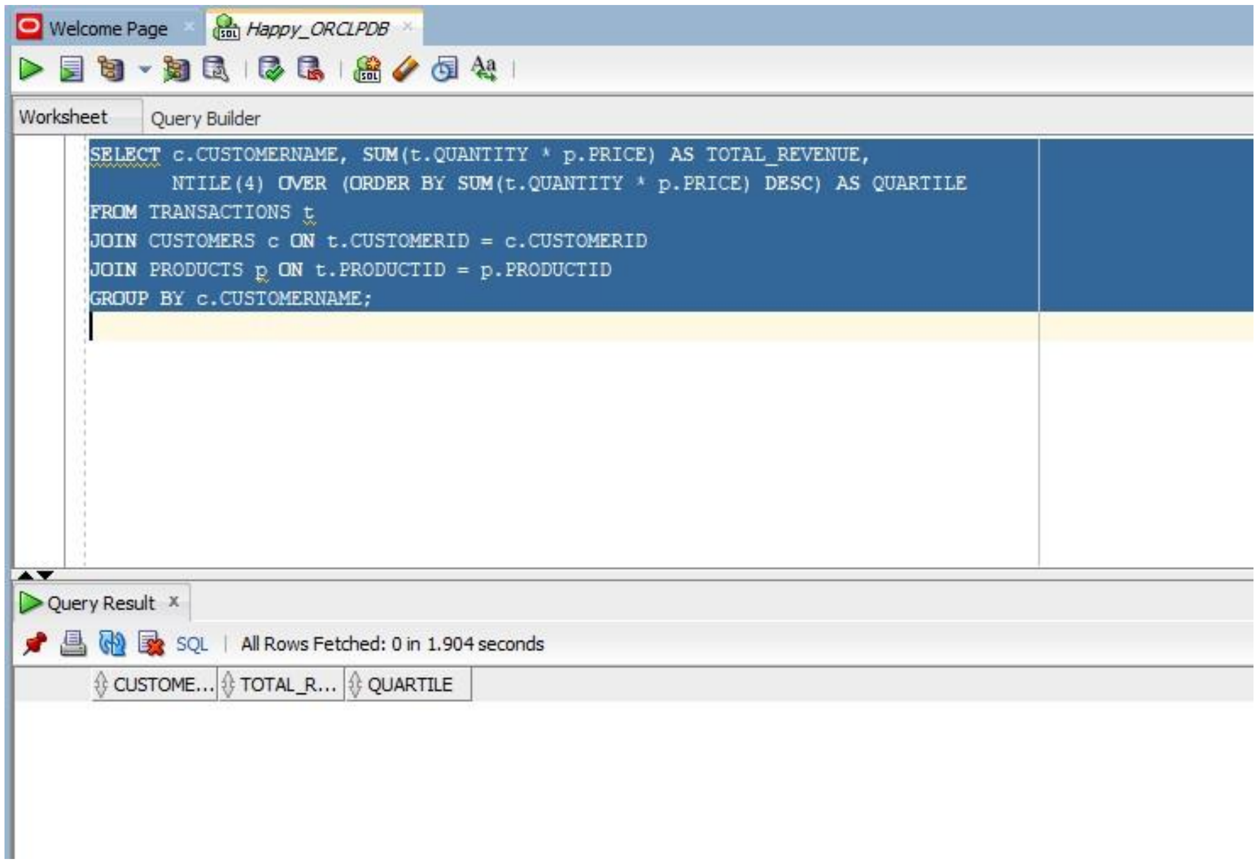
```
SELECT t.TRANSACTIONID, c.CUSTOMERNAME, t.TRANSACTIONDATE,
       (t.QUANTITY * p.PRICE) AS REVENUE,
       LAG(t.QUANTITY * p.PRICE) OVER (ORDER BY t.TRANSACTIONDATE) AS PREVIOUS_REVENUE
FROM TRANSACTIONS t
JOIN CUSTOMERS c ON t.CUSTOMERID = c.CUSTOMERID
JOIN PRODUCTS p ON t.PRODUCTID = p.PRODUCTID;
```

All Rows Fetched: 0 in 0.035 seconds

TRANSAC...	CUSTOME...	TRANSAC...	REVENUE	PREVIOU...

Measures growth or decline between consecutive transactions

Distribution Functions: Customer segmentation



Welcome Page | Happy_ORCLPDB

Worksheet | Query Builder

```sql
SELECT c.CUSTOMERNAME, SUM(t.QUANTITY * p.PRICE) AS TOTAL_REVENUE,
       NTILE(4) OVER (ORDER BY SUM(t.QUANTITY * p.PRICE) DESC) AS QUARTILE
FROM TRANSACTIONS t
JOIN CUSTOMERS c ON t.CUSTOMERID = c.CUSTOMERID
JOIN PRODUCTS p ON t.PRODUCTID = p.PRODUCTID
GROUP BY c.CUSTOMERNAME;
```

Query Result

SQL | All Rows Fetched: 0 in 1.904 seconds

CUSTOME...	TOTAL_R...	QUARTILE

Segments customers into quartiles for marketing strategies.