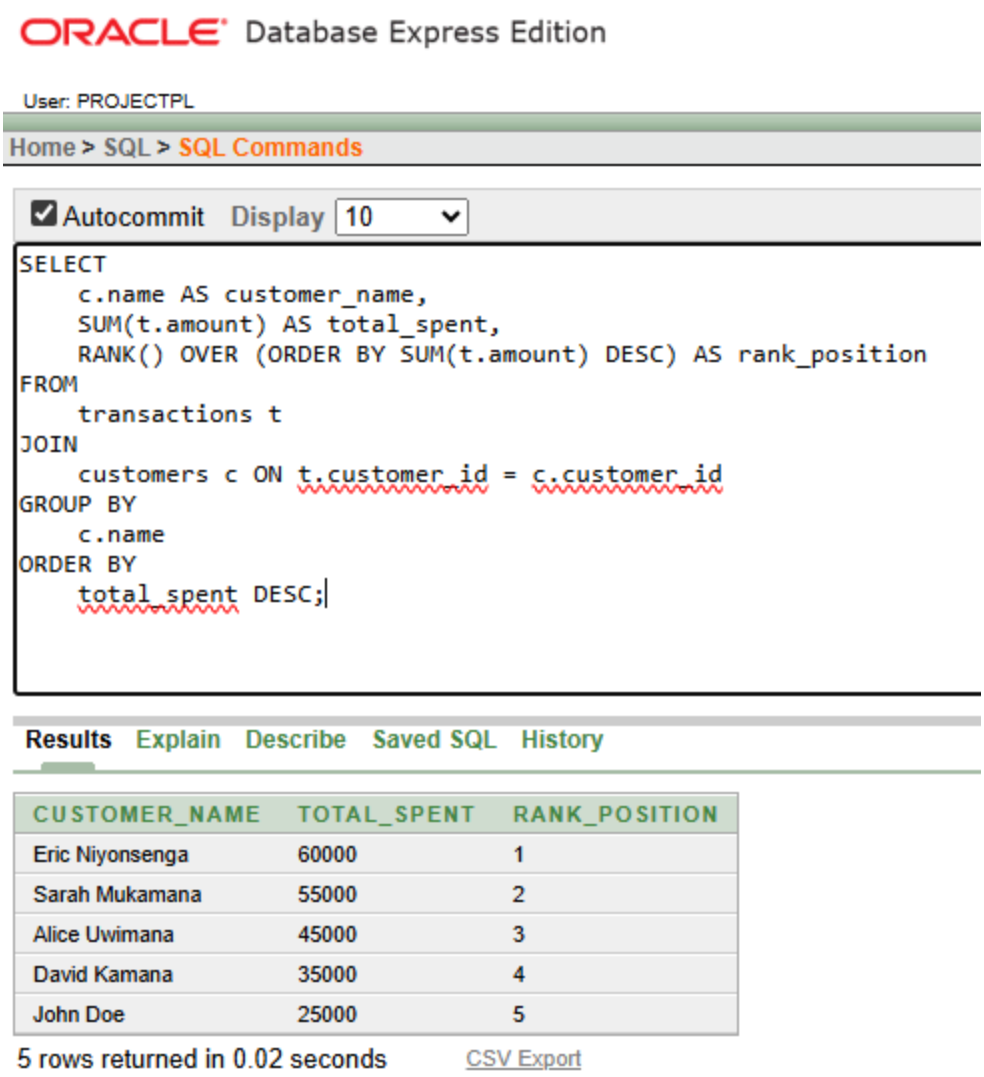
Ranking Functions

```
-- RANK() Window Function: Finds top customers based on total amount spent
SELECT
   c.name AS customer_name,
   SUM(t.amount) AS total_spent,
   -- RANK() assigns a rank based on total_spent (1 = highest spender)
   RANK() 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
ORDER BY
   total_spent DESC;
```

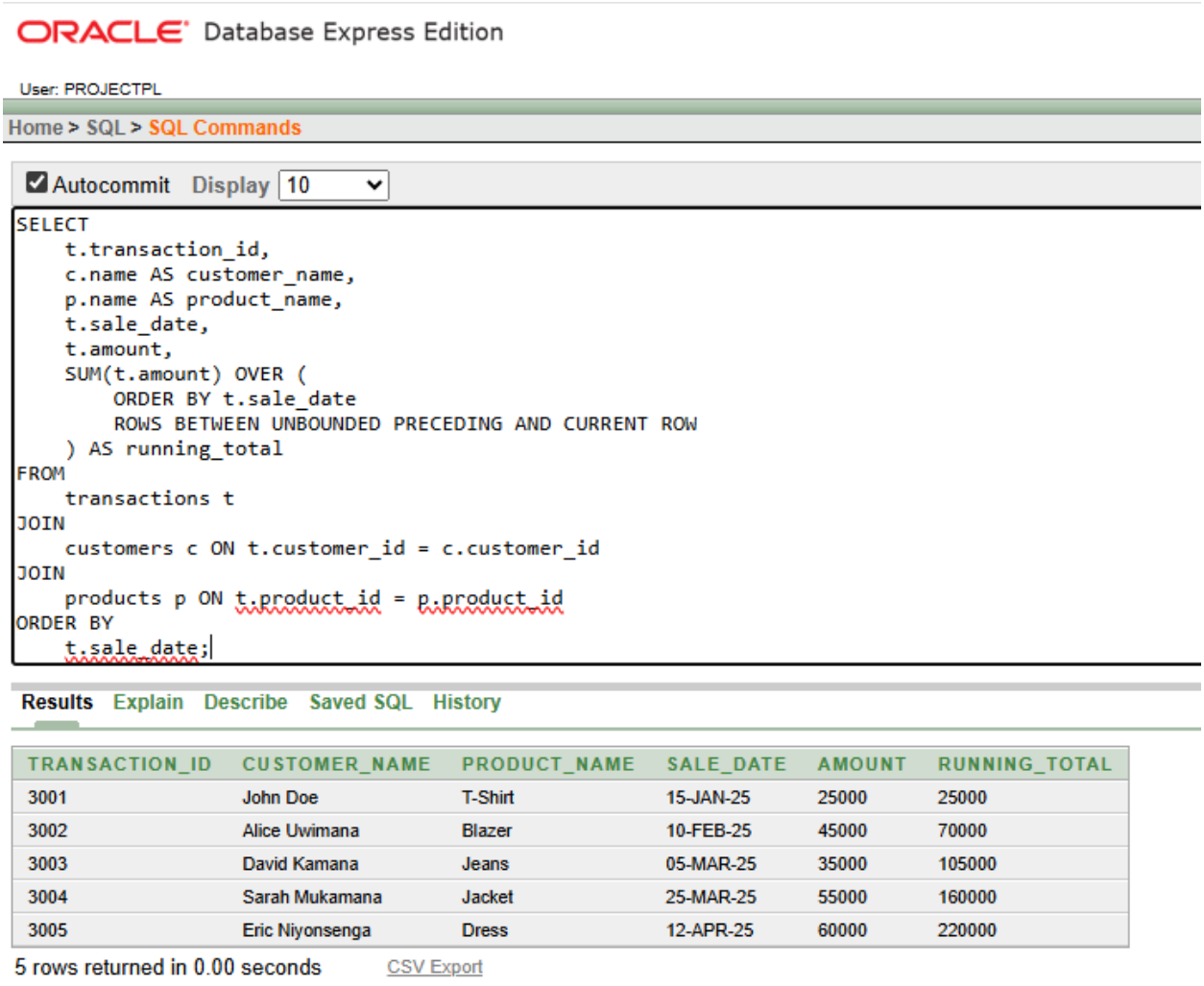


Interpretation:

This query shows which customers spent the most money in total. By ranking them, we can quickly identify our top-performing customers and focus marketing or loyalty programs on them. It helps the business understand who contributes most to revenue.

Aggregate Functions

```
-- SUM() OVER(): Calculates running total of sales over time
SELECT
   t.transaction_id,
   c.name AS customer_name,
   p.name AS product_name,
   t.sale_date,
   t.amount,
   -- Calculates cumulative total from the first row up to the current row
   SUM(t.amount) OVER (
      ORDER BY t.sale_date
      ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
   ) AS running_total
FROM
   transactions t
JOIN
   customers c ON t.customer_id = c.customer_id
JOIN
   products p ON t.product_id = p.product_id
ORDER BY
   t.sale_date;
```

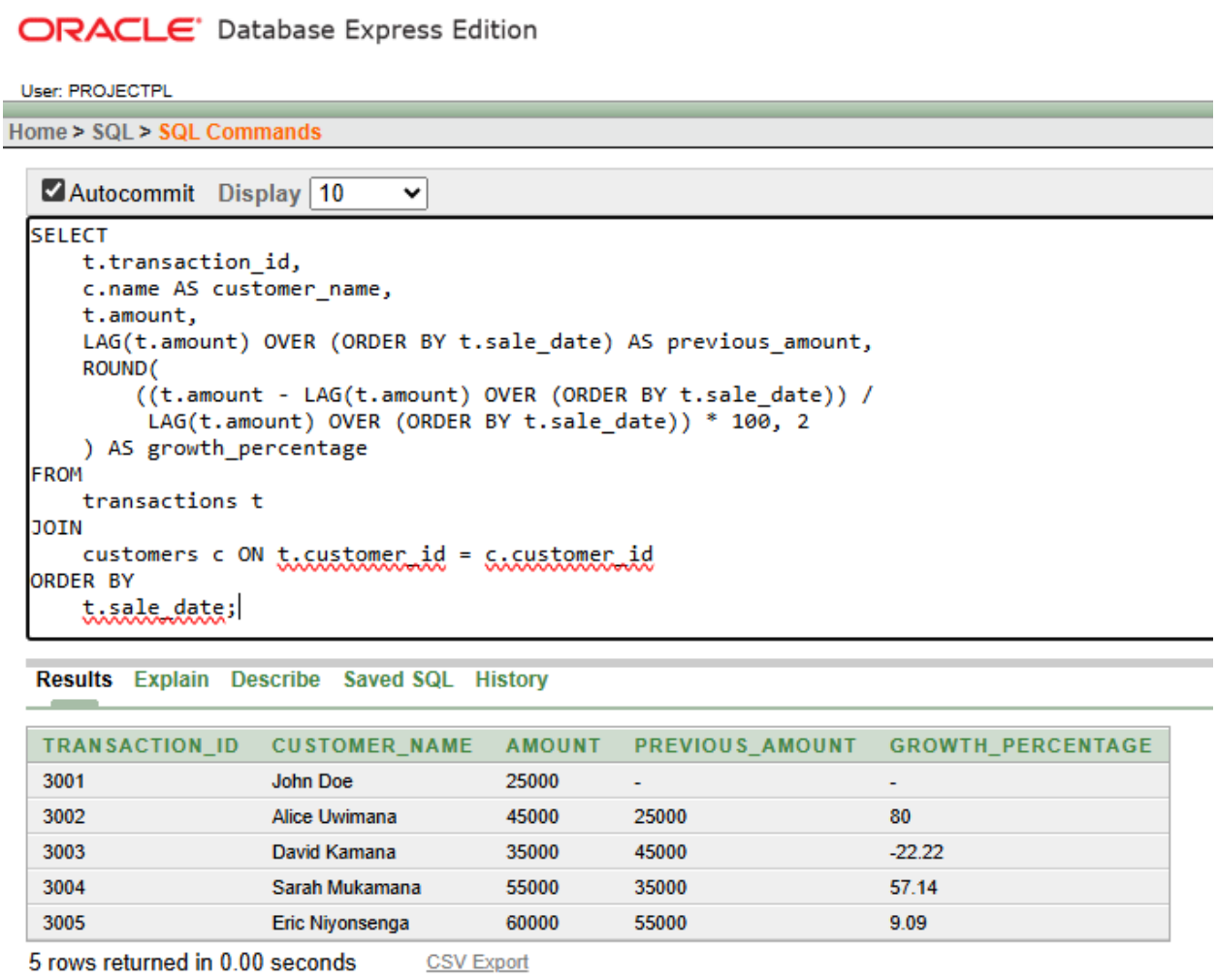


Interpretation:

The running total gives a clear view of how sales are growing over time. It shows cumulative revenue, which helps track performance and spot trends in monthly or quarterly income. This insight is useful for forecasting future sales and setting financial targets.

Navigation Functions

```
-- LAG(): Compares current transaction with the previous one
SELECT
   t.transaction_id,
   c.name AS customer_name,
   t.amount,
   -- LAG() fetches the previous transaction's amount
   LAG(t.amount) OVER (ORDER BY t.sale_date) AS previous_amount,
   -- Calculates percentage growth compared to previous transaction
   ROUND(
      ((t.amount - LAG(t.amount) OVER (ORDER BY t.sale_date)) /
       LAG(t.amount) OVER (ORDER BY t.sale_date)) * 100, 2
   ) AS growth_percentage
FROM
   transactions t
JOIN
   customers c ON t.customer_id = c.customer_id
ORDER BY
   t.sale_date;
```

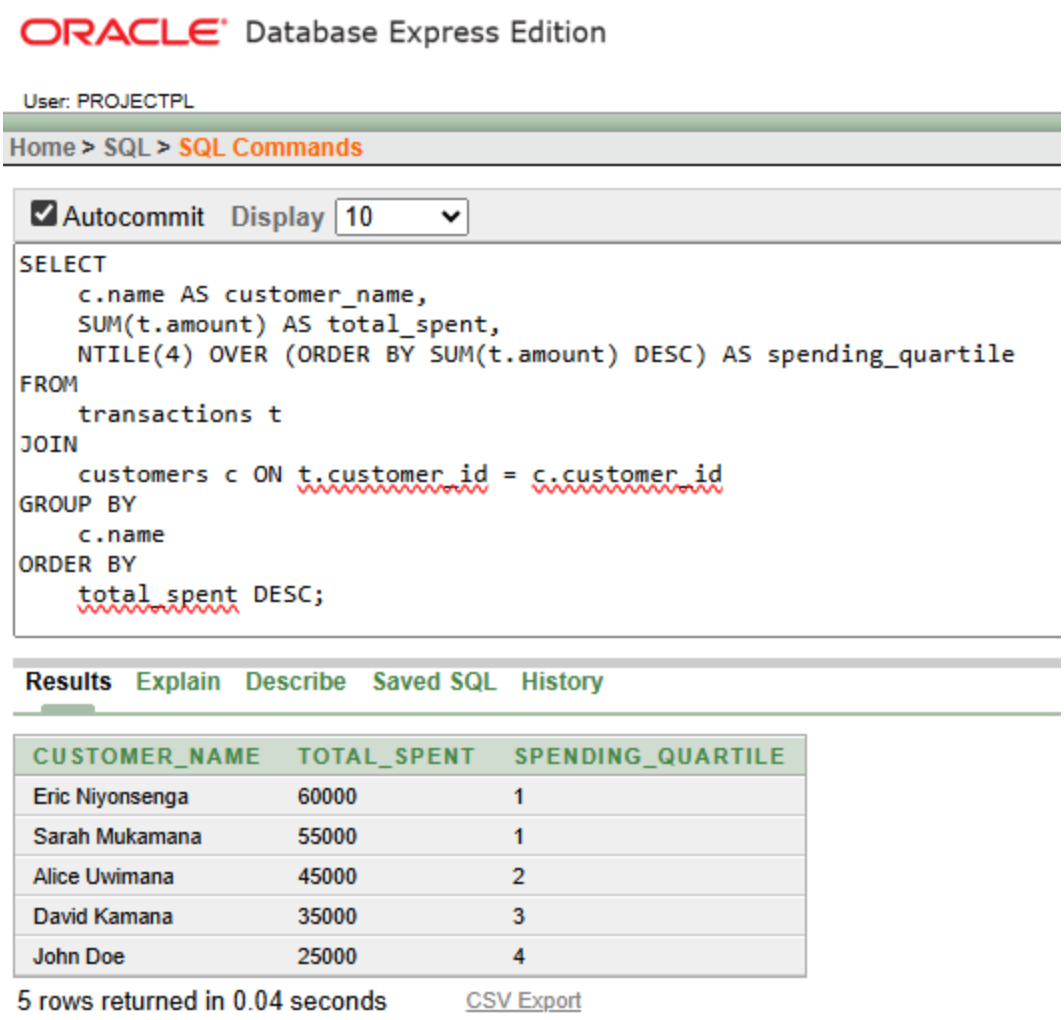


Interpretation:

This query calculates how much each transaction has grown or declined compared to the previous one. It reveals patterns in customer spending and helps identify months with strong or weak performance. Businesses can use this insight to adjust their strategies accordingly.

Distribution Functions

```
-- NTILE(4): Segments customers into 4 spending groups (quartiles)
SELECT
   c.name AS customer_name,
   SUM(t.amount) AS total_spent,
   -- NTILE(4) divides customers into 4 groups based on spending
   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
ORDER BY
   total_spent DESC;
```

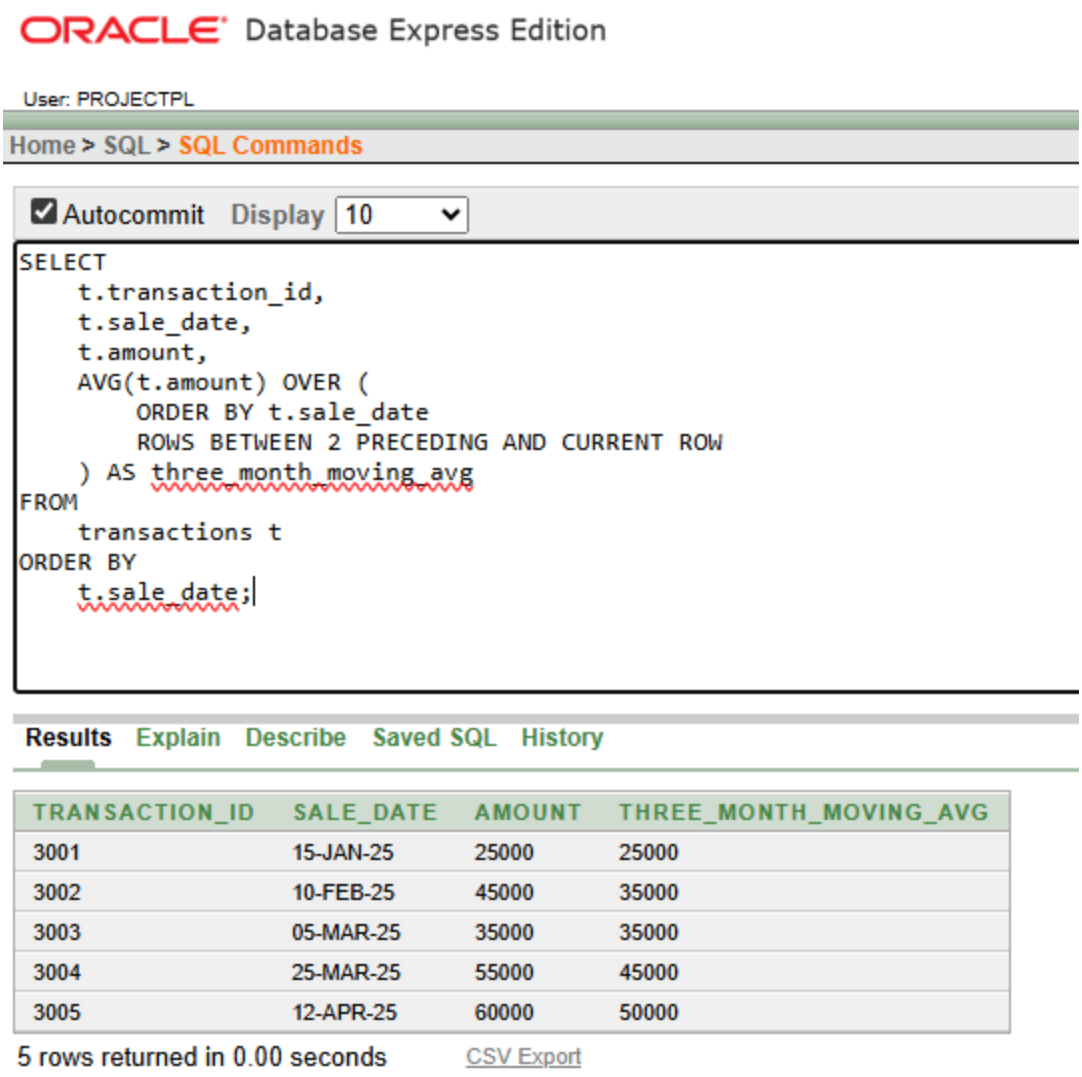


Interpretation:

This query divides customers into four groups based on their total spending. It helps the company segment customers (e.g., VIPs, regulars, occasional buyers) and design personalized marketing campaigns. Knowing which group each customer belongs to supports better decision-making.

Moving Average

```
-- AVG() OVER(): Calculates a 3-transaction moving average
SELECT
   t.transaction_id,
   t.sale_date,
   t.amount,
   -- Looks at the current row and 2 previous rows to compute average
   AVG(t.amount) OVER (
      ORDER BY t.sale_date
      ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
   ) AS three_month_moving_avg
FROM
   transactions t
ORDER BY
   t.sale_date;
```



Interpretation:

The moving average smooths out short-term fluctuations and highlights longer-term sales trends. It helps the business understand overall performance without being affected by one-time spikes or drops. This is valuable for strategic planning and forecasting.