Exercise 1: Analyzing Monthly Sales Trends with LAG in BigQuery

Objective

Learn to use the LAG function and OVER clause in SQL to analyze monthly sales trends in BigQuery.

Understanding the LAG Function and OVER Clause with Sales Data

The LAG Function

The LAG function is like looking back in a timeline of your sales records. It enables you to see the sales figure from the previous month, essential for comparing and understanding trends.

The OVER Clause – Organizing Data Chronologically

The OVER clause sets the order for your data, crucial for the LAG function to work correctly. By organizing the sales data in chronological order (using `ORDER BY year_month`), LAG correctly identifies the sales figure from the preceding month.

Example

- January Sales: $100,000 –> NONE
- February Sales: $120,000 –> January Sales: $100,000
- March Sales: $110,000 –> February Sales: $120,000

When analyzing March, `LAG(total_sales) OVER (ORDER BY year_month)` fetches February's sales, allowing you to calculate the growth rate from February to March.

Step-by-Step Guide to Building a Monthly Sales Trend Analysis Query

Step 1: Calculating Total Monthly Sales

Task: Write a query to calculate the total sales for each month.

Syntax and Example:

```
SELECT
  FORMAT_TIMESTAMP('%Y-%m', orderdate) AS year_month,
  SUM(CAST(totaldue AS FLOAT64)) AS total_sales
```

```
FROM
  [Your Table]
GROUP BY year_month
```

Skeleton:

```
SELECT
  /* Use FORMAT_TIMESTAMP to format orderdate as year_month */
  /* Use SUM to calculate total sales */
FROM
  `adventureworks.dwh_sales_data`
/* Remember to GROUP BY year_month */
```

Step 2: Retrieving Previous Month's Sales

Task: Use the LAG function to fetch the total sales of the previous month.

Syntax and Example:

```
LAG(column_name) OVER (ORDER BY column_name)
```

Skeleton:

```
WITH MonthlySales AS (
  /* Insert your Step 1 query here */
)
SELECT
  year_month,
  total_sales,
  /* Use LAG to retrieve previous month's sales */
FROM
  MonthlySales
```

Step 3: Calculating the Growth Rate

Task: Calculate the month-over-month growth rate in sales.

Syntax and Example:

```
ROUND(((current_value - previous_value) / previous_value) * 100, 2)
```

Skeleton:

```
/* Include your full query from Step 2 here */
SELECT
```

```
  year_month,
  total_sales,
  previous_month_sales,
  /* Calculate growth rate using the provided syntax */
FROM
  /* Your source table from Step 2 */
```

Step 4: Putting It All Together

Skeleton for the Complete Query:

```
WITH MonthlySales AS (
  /* Insert your query from Step 1 here */
)
SELECT
  year_month,
  /* Insert total_sales calculation */
  /* Insert previous_month_sales calculation using LAG */
  /* Insert growth rate calculation */
FROM
  MonthlySales
ORDER BY
  year_month;
```

Conclusion

This exercise will help you understand and apply the LAG function and OVER clause in SQL, enabling you to analyze trends in monthly sales data. The step-by-step approach and simplified templates are designed to make the learning process more accessible and enjoyable.

Exercise 2: Segmenting Customers Based on Spending Percentiles

Objective

Learn to use the `PERCENTILE_CONT` function in SQL to analyze customer spending patterns and segment customers based on their spending percentiles in BigQuery.

Further Reading on Percentiles

For a more in-depth understanding of percentiles and their applications in statistics, you can refer to the Wikipedia article on the topic: Percentile - Wikipedia.

Understanding PERCENT_RANK and the OVER Clause

What is PERCENT_RANK?

PERCENT_RANK is a window function in SQL used to calculate the percentile rank of each row within a partition of a result set. The percentile rank indicates the relative standing of a value within the dataset.

How Does PERCENT_RANK Work?

- The function assigns a rank to each row, with the lowest value getting a rank of 0.
- The highest value receives a rank of 1.
- Intermediate values are assigned a percentile rank that reflects their position in the ordered set.

Syntax of PERCENT_RANK

```
PERCENT_RANK() OVER (ORDER BY column_name)
```

The OVER Clause

- The OVER clause defines the window (or set of rows) over which the PERCENT_RANK function operates.
- It determines how the data is partitioned and ordered before the function is applied.
- The ORDER BY part of the OVER clause is crucial as it defines the order in which the data is ranked.

Choosing the Right Column for OVER

- The column you choose for the ORDER BY part of the OVER clause should reflect the metric you're interested in ranking.
- In the context of customer spending, this would typically be a column representing the amount spent, total sales, or similar financial metrics.

Step-by-Step Guide to Building a Customer Spending Segmentation Query

Step 1: Calculating Total Spending per Customer

Task: Write a query to calculate the total spending for each customer.

Hints: - Use the SUM function to calculate the total spending. - Multiply `unitprice` by `orderqty` for each order. - Group the results by `customerid`.

Skeleton:

```
SELECT
  customerid,
  /* Complete this line */
FROM
  `adventureworks.dwh_sales_data`
/* Complete this line */
```

Step 2: Ranking Customers by Spending

Task: Once you have total spending per customer, rank these customers based on their spending.

Hints: - Use the PERCENT_RANK function to assign a percentile rank. - Order the results by total spending.

Skeleton:

```
WITH CustomerSpending AS (
  /* Insert your Step 1 query here */
)
SELECT
  /* Complete this line for PERCENT_RANK */
  customerid,
  total_spending
FROM
  CustomerSpending
```

Step 3: Categorizing Customers Based on Spending Percentiles

Task: Use the percentile ranks to categorize customers into spending groups.

Hints: - Use a CASE statement to assign categories. - Define categories such as 'Low Spender', 'Medium Spender', etc., based on percentile ranks.

Skeleton:

```
WITH
  CustomerSpending AS (
    /* Insert your Step 1 query here */
  ),
  PercentileRanks AS (
    /* Insert your Step 2 query here */
  )
SELECT
  /* Complete the CASE statement */
FROM
  PercentileRanks
/* Remember to add the GROUP BY clause */
```

Step 4: Counting Customers in Each Category

Task: Count the number of customers in each spending category.

Hints: - Use `COUNT(*)` to count customers. - Group the results by the spending category.

Skeleton:

```
/* Include your full query from Step 3 here */
GROUP BY
  /* Complete this line */
ORDER BY
  /* Complete this line */
```

Step 5: Putting It All Together

Skeleton for the Complete Query

```
WITH
  CustomerSpending AS (
    /* Insert your query from Step 1 here */
  ),
  PercentileRanks AS (
    /* Insert your query from Step 2 here */
  )
SELECT
  /* Insert your CASE statement from Step 3 here */
  /* Insert the COUNT function from Step 4 here */
FROM
```

```
  /* Insert your source from Step 3 here */
GROUP BY
  /* Complete this line */
ORDER BY
  /* Complete this line */
```

Conclusion

This exercise provides a detailed understanding of the PERCENT_RANK function and the OVER clause, essential tools in SQL for analyzing data distributions and rankings. By completing this exercise, you'll gain practical skills in constructing complex SQL queries for real-world data analysis scenarios.