**LAG( )** window function is used to retrieve data from the preceding row in a specified order.

## BASIC LAG:

**1.** SELECT * FROM sales;

| months | revenue |
|---|---|
| January | 1000 |
| March | 1500 |
| February | 1200 |
| April | 1300 |

### Retrieve the monthly revenue along with the revenue of the previous month using the LAG window function.

**Query :**

```
SELECT months, revenue,
        LAG(revenue) OVER (ORDER BY months) AS prev_months_revenue
FROM sales;
```

**Output :**

| months | revenue | prev_months_revenue |
|---|---|---|
| April | 1300 | NULL |
| February | 1200 | 1300 |
| January | 1000 | 1200 |
| March | 1500 | 1000 |

## LAG with Custom Partition:

**2.** SELECT * FROM sales_data;

| months | revenue |
|---|---|
| January | 1300 |
| February | 1200 |
| January | 1000 |
| March | 1500 |
| January | 1220 |
| February | 1100 |
| April | 1300 |

**Retrieve the monthly revenue along with the revenue of the previous month using the LAG window function.**

**Query :**

```
SELECT months, revenue,
        LAG(revenue) OVER (PARTITION BY months ORDER BY revenue) AS prev_months_revenue
FROM sales_data;
```

**Output :**

| months | revenue | prev_months_revenue |
|---|---|---|
| April | 1300 | NULL |
| February | 1100 | NULL |
| February | 1200 | 1100 |
| January | 1000 | NULL |
| January | 1220 | 1000 |
| January | 1900 | 1220 |
| March | 1500 | NULL |

## 3. LAG with Default Value :

Retrieve the revenue for each month along with the revenue for the previous month, using a default value of 5 for the first month

**Query:**

```
SELECT months, revenue,
        LAG(revenue, 1, 5) OVER (ORDER BY
        CASE months
            WHEN 'January' THEN 1
            WHEN 'February' THEN 2
            WHEN 'March' THEN 3
            WHEN 'April' THEN 4
            ELSE NULL
        END
    ) AS prev_month_revenue_default
FROM  sales_data;
```

**Output :**

| months | revenue | prev_month_revenue_default |
|---|---|---|
| January | 1000 | 5 |
| January | 1900 | 1000 |
| January | 1220 | 1900 |
| February | 1200 | 1220 |

| | | |
|---|---|---|
| February | 1100 | 1200 |
| March | 1500 | 1100 |
| April | 1300 | 1500 |

**4.** SELECT * FROM covid;

| city | days | cases |
|---|---|---|
| DELHI | 2022-01-01 | 100 |
| DELHI | 2022-01-02 | 200 |
| DELHI | 2022-01-03 | 300 |
| MUMBAI | 2022-01-01 | 100 |
| MUMBAI | 2022-01-02 | 100 |
| MUMBAI | 2022-01-03 | 300 |
| CHENNAI | 2022-01-01 | 100 |
| CHENNAI | 2022-01-02 | 200 |
| CHENNAI | 2022-01-03 | 150 |
| BANGALORE | 2022-01-01 | 100 |
| BANGALORE | 2022-01-02 | 300 |
| BANGALORE | 2022-01-03 | 200 |
| BANGALORE | 2022-01-04 | 400 |

## Find cities where the covid cases are increasing continuous

```
WITH cte AS (
   SELECT city, days, cases,
        LAG(cases, 1, 0) OVER (PARTITION BY city ORDER BY days) AS pre_cases
   FROM covid
)
SELECT city
FROM cte
GROUP BY city
HAVING SUM(CASE WHEN cases <= pre_cases THEN 1 ELSE 0 END) = 0
ORDER BY city;
```

**Output:**

DELHI