PostgreSQL LEAD Function

Summary: in this tutorial, you will learn how to use the PostgreSQL `LEAD()` function to access a row that follows the current row, at a specific physical offset.

Introduction to PostgreSQL LEAD() function

PostgreSQL `LEAD()` function provide access to a row that follows the current row at a specified physical offset.

It means that from the current row, the `LEAD()` function can access data of the next row, the row after the next row, and so on.

The `LEAD()` function is very useful for comparing the value of the current row with the value of the row that following the current row.

The following illustrates the syntax of `LEAD()` function:

```
LEAD(expression [,offset [,default_value]])
OVER (
    [PARTITION BY partition_expression, ... ]
    ORDER BY sort_expression [ASC | DESC], ...
)
```

In this syntax:

expression

The `expression` is evaluated against the following row based on a specified offset from the current row. The `expression` can be a column, expression, subquery (https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-subquery/) that must evaluate to a single value. And it cannot be a window function (https://www.postgresqltutorial.com/postgresql-

window-function/) .

offset

The `offset` is a positive integer that specifies the number of rows forwarding from the current row from which to access data. The `offset` can be an expression, subquery, or column.

The offset defaults to 1 if you don't specify it.

default_value

The `default_value` is the return value if the `offset` goes beyond the scope of the partition. The `default_value` defaults to NULL if you omit it.

PARTITION BY clause

The `PARTITION BY` clause divides rows into partitions to which the `LEAD()` function is applied.

By default, the whole result set is a single partition if you omit the `PARTITION BY` clause.

ORDER BY clause

The `ORDER BY` clause specifies the sort order of the rows in each partition to which the `LEAD()` function is applied.

PostgreSQL LEAD() function examples

Let's set up a new table for the demonstration.

First, create a new table (https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-create-table/) named `sales` :

```
CREATE TABLE sales(
        year SMALLINT CHECK(year > 0),
        group_id INT NOT NULL,
```

```
        amount DECIMAL(10,2) NOT NULL,
        PRIMARY KEY(year,group_id)
);
```

Second, insert (https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-insert/) some rows into the `sales` table:

```
INSERT INTO
        sales(year, group_id, amount)
VALUES
        (2018,1,1474),
        (2018,2,1787),
        (2018,3,1760),
        (2019,1,1915),
        (2019,2,1911),
        (2019,3,1118),
        (2020,1,1646),
        (2020,2,1975),
        (2020,3,1516);
```

Third, query data from the `sales` table:

```
SELECT * FROM sales;
```

1) Using PostgreSQL LEAD() function over a result set examples

The following query returns the total sales amount by year:

```
SELECT
        year,
        SUM(amount)
FROM sales
GROUP BY year
ORDER BY year;
```

This example uses the `LEAD()` function to return the sales amount of the current year and the next year:

```
WITH cte AS (
        SELECT
                year,
                SUM(amount) amount
        FROM sales
        GROUP BY year
        ORDER BY year
)
SELECT
        year,
        amount,
        LEAD(amount,1) OVER (
                ORDER BY year
        ) next_year_sales
FROM
```

```
        cte;
```

Here is the output:

In this example:

- First, the CTE (https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-cte/) returns the sales summarized by year.
- Then, the outer query uses the `LEAD()` function to return the sales of the following year for each row.

The following example uses two common table expressions to return the sales variance between the current year and the next:

```
WITH cte AS (
        SELECT
                year,
                SUM(amount) amount
        FROM sales
        GROUP BY year
        ORDER BY year
), cte2 AS (
        SELECT
                year,
                amount,
                LEAD(amount,1) OVER (
                        ORDER BY year
                ) next_year_sales
        FROM
                cte
```

```
)
SELECT
        year,
        amount,
        next_year_sales,
        (next_year_sales - amount) variance
FROM
        cte2;
```

2) Using PostgreSQL LEAD() function over a partition example

The following statement uses the `LEAD()` function to compare the sales of the current year with sales of the next year for each product group:

```
SELECT
        year,
        amount,
        group_id,
        LEAD(amount,1) OVER (
                PARTITION BY group_id
                ORDER BY year
        ) next_year_sales
FROM
        sales;
```

This picture shows the output:

In this example:

- The `PARTITION BY` clause distributes rows into product groups (or partitions) specified by group id.
- The `ORDER BY` clause sorts rows in each product group by years in ascending order.
- The `LEAD()` function returns the sales of the next year from the current year for each product group.

In this tutorial, you have learned how to use the PostgreSQL `LEAD()` function to access a row at a specific physical offset which follows the current row.