Limiting Results with Window Functions

Jason P. Browning
ASST. COMMISSIONER OF HIGHER EDUCATION, STATE OF MISSOURI

@jason_from_ky

Overview

Window function syntax

Applying window functions

Aggregate functions as window functions

Filtering window function results

Defining the Window

A **window** is a set of table rows over which the function is applied

The rows must be somehow related

Window Function Syntax

OVER()

The OVER() clause defines the set of rows to which the function will be applied

PARTITION BY

Rows are partitioned to form groups of rows

ORDER BY

Rows within each partition are ordered

Window Functions

ROW_NUMBER()

RANK()

DENSE_RANK()

FIRST_VALUE()

LAST_VALUE()

LAG()

LEAD()

```
SELECT
   name,
   course,
   ROW_NUMBER() OVER () AS rn
  FROM enrollees;
```

Numbering Rows

ROW_NUMBER() window function assigns a sequential number to each row

The window is defined in the OVER clause; if not defined, the entire result set becomes the window

```
SELECT
   name,
   course,
   ROW_NUMBER() OVER () AS rn
 FROM enrollees;
```

name	course	rn
Jason	Accounting I	1
Lucy	Health Science	2
Martha	Biology	3
Lucy	Architecture	4
Jason	Economics	5

Numbering Rows

ROW_NUMBER function assigns a sequential number to each row

The window is defined in the OVER clause

If not defined, the entire result set becomes the window

```
SELECT
   name,
   course,
   ROW_NUMBER() OVER (
     PARTITION BY name
     ORDER BY course) AS rn
  FROM enrollees;
```

Numbering Rows

- ROW_NUMBER() window function assigns a sequential number to each row
- The window is defined in the OVER clause
- The window function resets for each partition defined using the PARTITION BY keyword
- Within each partition, rows are ordered based on the order specified using the ORDER BY keyword

```
SELECT
  name,
  course,
  ROW_NUMBER() OVER (
    PARTITION BY name
    ORDER BY course) AS rn
FROM enrollees;
```

Numbering Rows

name	course
Jason	Accounting I
Lucy	Health Science
Martha	Biology
Lucy	Architecture
Jason	Economics

name	course	rn
Jason	Accounting I	1
Jason	Economics	2
Lucy	Architecture	1
Lucy	Health Science	2
Martha	Biology	1

Ranking Rows

RANK() assigns a sequential row number

Behaves similarly to ROW_NUMBER(), except matching rows are ranked the same

Use DENSE_RANK() to avoid gaps in ranking

```
SELECT name,
       RANK() OVER (
         ORDER BY name) AS rank
  FROM enrollees;
```

name	rank
Jason	1
Jason	1
Lucy	3
Lucy	3
Martha	5

Ranking Rows

RANK() function assigns a ranking to each row

Matching rows are equally ranked (tied)

```
SELECT name,
      RANK() OVER (
        ORDER BY name) AS rank,
      DENSE_RANK() OVER (
        ORDER BY name) AS d_rank,
      ROW_NUMBER() OVER (
        ORDER BY name) AS rn
  FROM enrollees;
```

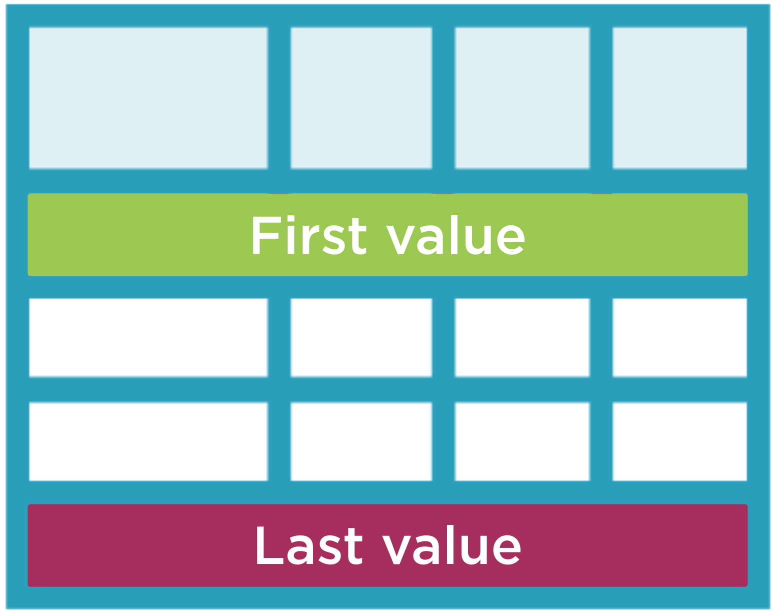
- RANK() function assigns a ranking to each row
- Matching rows are equally ranked (tied)
- To avoid gaps in ranking, use DENSE_RANK()

name	rank	d_rank	rn
Jason	1	1	1
Jason	1	1	2
Lucy	3	2	3
Lucy	3	2	4
Martha	5	3	5

Demo

Ranking rows using window functions

Special Values



FIRST_VALUE() returns the first value in an ordered set of values

LAST_VALUE() returns the last value in an ordered set of values

```
SELECT dept,
       FIRST_VALUE(name) OVER (
         PARTITION BY dept
         ORDER BY salary DESC)
        AS high_pay
  FROM salaries;
```

- **FIRST_VALUE()** returns the first value in an ordered set of values

name	dept	salary
June	HR	$85,000
Alice	HR	$40,000
John	Finance	$95,000
Kim	Finance	$82,000
Eunice	Finance	$90,000
Alex	HR	$88,000

dept	high_pay
Finance	John
HR	Alex

```
SELECT dept,
     LAST_VALUE(name) OVER (
       PARTITION BY dept
       ORDER BY salary DESC)
       AS low_pay,
     LAST_VALUE(salary) OVER (
       PARTITION BY dept
       ORDER BY salary DESC)
       AS low_amt
  FROM salaries;
```

◀ LAST_VALUE() returns the last value in an ordered set of values

name	dept	salary
June	HR	$85,000
Alice	HR	$40,000
John	Finance	$95,000
Kim	Finance	$82,000
Eunice	Finance	$90,000
Alex	HR	$88,000

dept	low_pay	low_amt
Finance	Kim	$82,000
HR	Alice	$40,000

Reusing the Window Definition

```
SELECT dept,
    LAST_VALUE(name) OVER (PARTITION BY dept ORDER BY salary DESC)
     AS low_pay,
    LAST_VALUE(salary) OVER (PARTITION BY dept ORDER BY salary DESC)
     AS low_amt
 FROM salaries;
```

```
SELECT dept,
    LAST_VALUE(name) OVER w AS low_pay,
    LAST_VALUE(salary) OVER w AS low_amt
 FROM salaries
WINDOW w AS (
     PARTITION BY dept
     ORDER BY salary DESC);
```

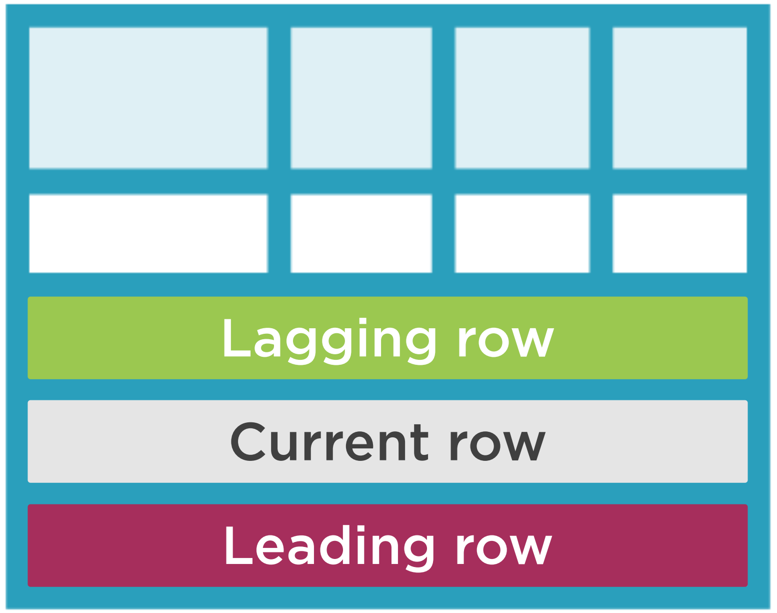
Reusing the Window Definition

Windows can be defined using the WINDOW clause

Reference the window definitions using the OVER keyword

WINDOW clause comes after WHERE clause and precedes ORDER BY

Lagging and Leading Rows



Lagging rows occur before the current row

Leading rows occur after the current row

The LAG() and LEAD() functions can be used to access these rows

```
SELECT month,
       sales AS curr_month,
        LAG(monthly_sales,1) OVER (
          ORDER BY month)
          AS prev_month
 FROM sales_record
ORDER BY month;
```

◀ Specify both the field to return and the number of rows to offset using the LAG() function

month	sales
1	$ 100,000
2	$ 76,000
3	$ 98,000

month	curr_month	prev_month
1	$ 100,000	NULL
2	$ 76,000	$ 100,000
3	$ 98,000	$ 76,000

Aggregate functions may also be applied as window functions.

Aggregate Functions

- COUNT
- SUM
- AVG
- MIN
- MAX

```
SELECT grade_lvl,
       AVG(age) AS avg_age
  FROM students
 GROUP BY grade_lvl;
```

grade_lvl	avg_age
Freshman	15
Junior	17
Senior	19

- Aggregate average function
- Group results by grade level

name	grade_lvl	age
Eliza	Junior	17
Jane	Junior	17
Leslie	Senior	19
Matt	Junior	16
Ned	Freshman	15
Susie	Junior	18

```
SELECT grade_lvl,
       AVG(age) OVER (
         PARTITION BY grade_lvl)
         AS avg_age
  FROM students;
```

- Aggregate average function
- To apply aggregate function as a window function, use OVER clause
- Partition results by grade level

name	grade_lvl	age	avg_age
Eliza	Junior	17	17
Jane	Junior	17	17
Leslie	Senior	19	19
Matt	Junior	16	17
Ned	Freshman	15	15
Susie	Junior	18	17

Order of Evaluation

Window functions are evaluated after joins, grouping, and having clauses

Window functions are evaluated at the same time as other SELECT statements

To use other selections in a window function

- include selection in a subquery or CTE
- apply window function in outer query

Window Functions in Evaluation

Window function may be used in calculations in SELECT statement

Window function may not be used in WHERE or HAVING clause

Use window function in subquery or CTE and access window function result using outer query for subsequent calculations

Using Window Functions to Filter Results

```
SELECT a.first_name,
       a.last_name
  FROM (SELECT first_name,
               last_name,
               ROW_NUMBER() OVER (
                 PARTITION BY dept
                 ORDER BY last_name)
                AS rn
          FROM  students) a
 WHERE rn = 1
```

Window functions cannot be used in WHERE or HAVING clauses

Function is evaluated after these clauses

Place function in subquery or CTE and use as limiting criteria in outer query

Demo

Using the LAG() window function

Using window functions in calculations

Summary

Window functions allow for calculations and analysis at various levels of detail

Aggregate functions can also be applied as window functions

Window function results cannot be used in the WHERE or HAVING clause