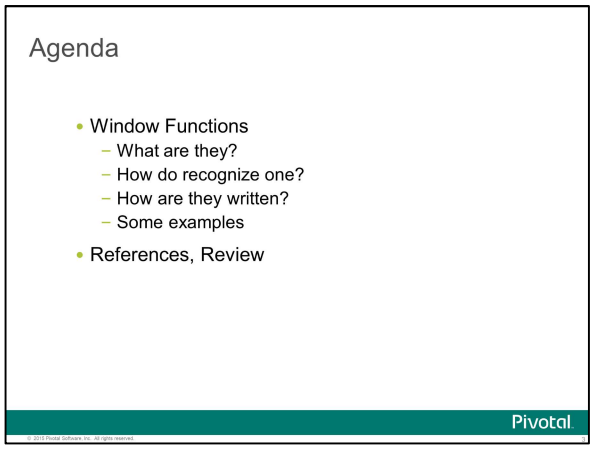

Pivotal
A NEW PLATFORM FOR A NEW ERA

# Window Functions

Pivotal® Greenplum Database

Pivotal



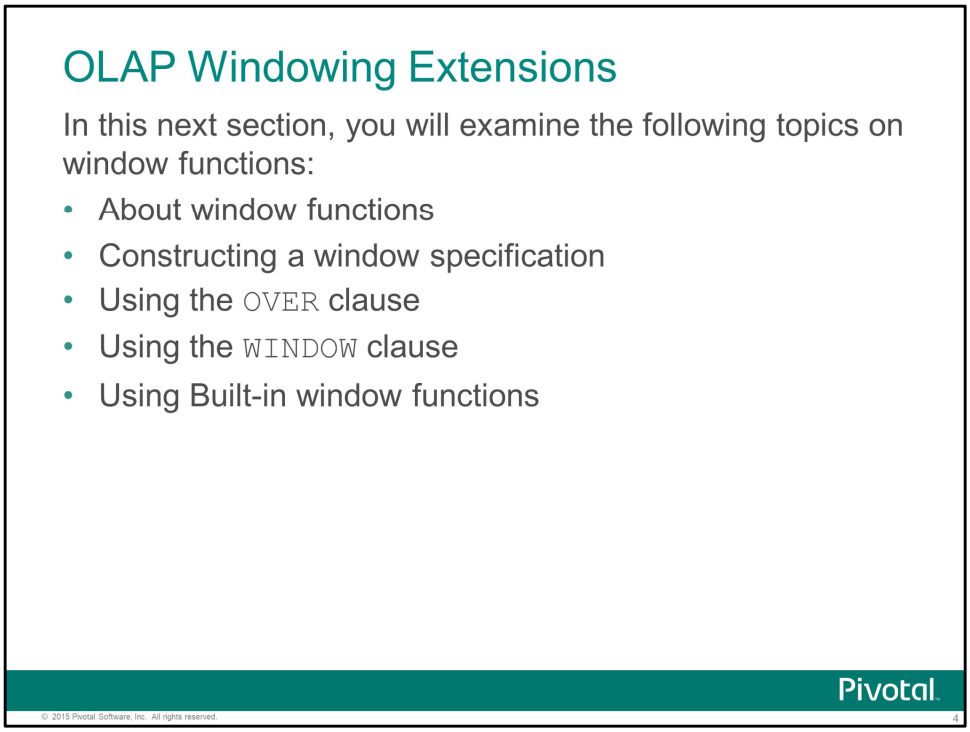

Hello, this is Marshall Presser. In this section, we'll be talking about how you can use some important functions that are part of the Pivotal Greenplum Database to help solve analytic business problems. Some come with the database itself and others are easily installed.

First, we'll consider set based operations and grouping operators.
Then we'll look at some built in functions and do a brief demo

Next we'll look at the machine learning functions that come with Madlib, a package of very useful functions that make doing Data Science in GPDB a lot easier

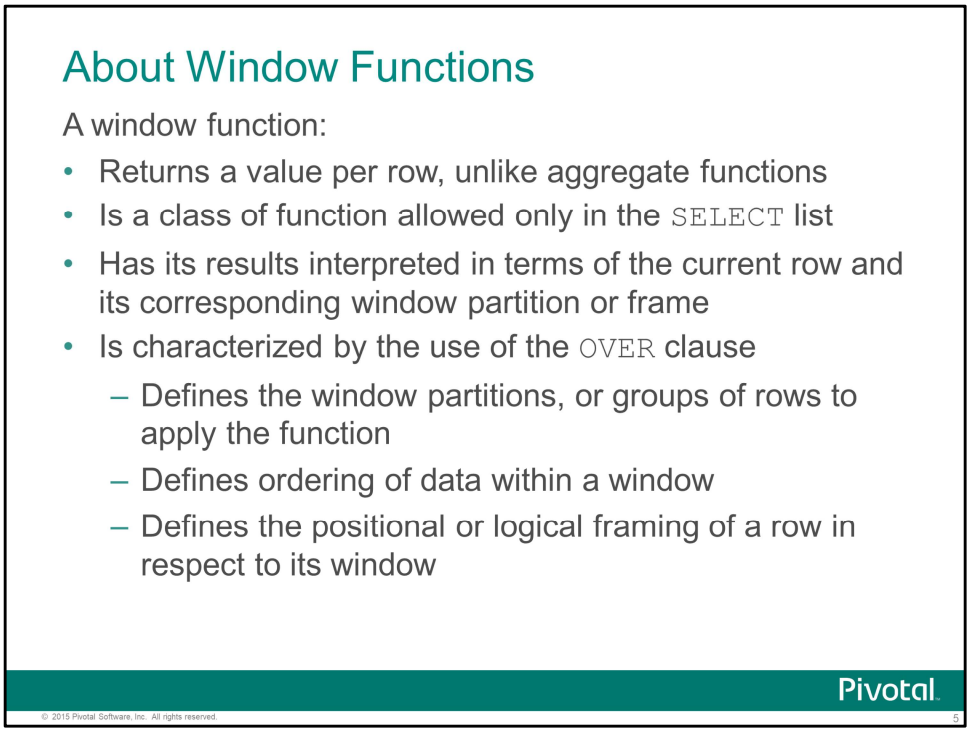
We'll also look at PostGIS, the PostgreSQL geospatial database functions and have another brief demo.

Lastly, we'll look at Window functions and do a demo arising from their use at one of our customers.




Greenplum supports the concept of OLAP analytical functions, also known as window functions. In this next section, you will examine:

- Key concepts of window functions and how they are used.
- The SQL syntax used to construct the window specification used by window functions:
- The use of the `OVER` clause, which is a window function classified by the use of the special OVER clause, used to define the window specification.
- The use of the `WINDOW` clause, which is a convenient feature for defining and naming window specifications that can be used in one or more window function `OVER` clauses of a query.
- The built-in window functions provided in Greenplum Database.




Window functions are a new class of functions introduced in Greenplum.

Window functions allow application developers to more easily compose complex OLAP queries using standard SQL commands. For example:

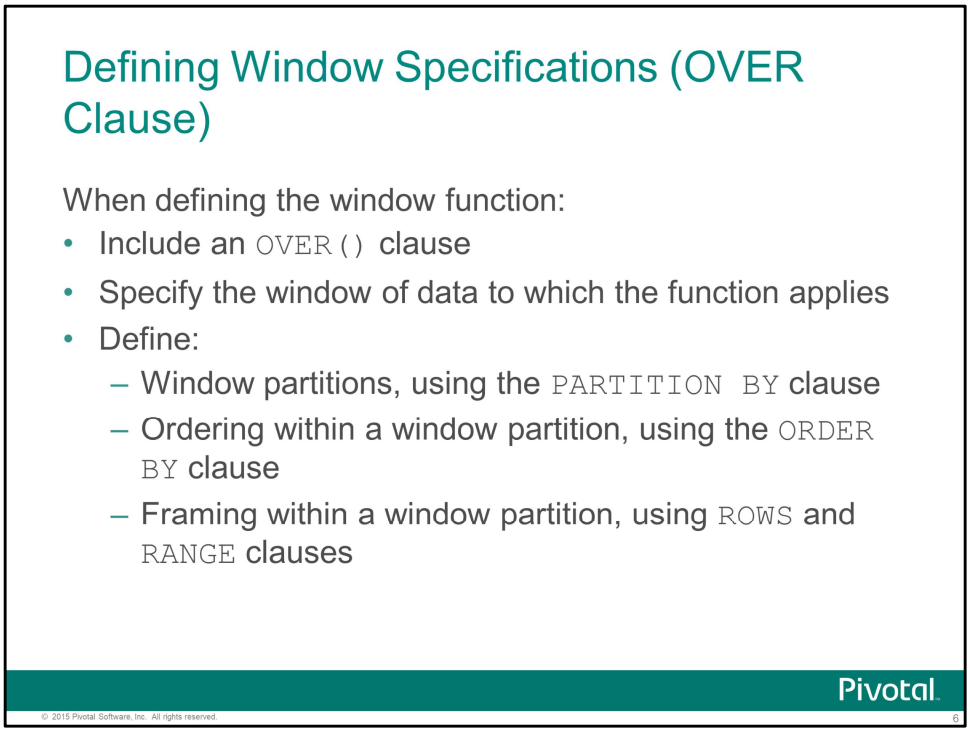
- Moving averages or sums can be calculated over various intervals.
- Aggregations and ranks can be reset as selected column values change.
- Complex ratios can be expressed in simple terms.

Window functions can only be used in the `SELECT` list, between the `SELECT` and `FROM` keywords of a query.

Unlike aggregate functions, which return a result value for each group of rows, window functions return a result value for every row, but that value is calculated with respect to the rows in a particular window partition (grouping) or window frame (row position within the window).

What classifies a function as a window function is the use of an `OVER` clause. The `OVER` clause defines the window of data to which the function will be applied. There are three characteristics of a window specification:

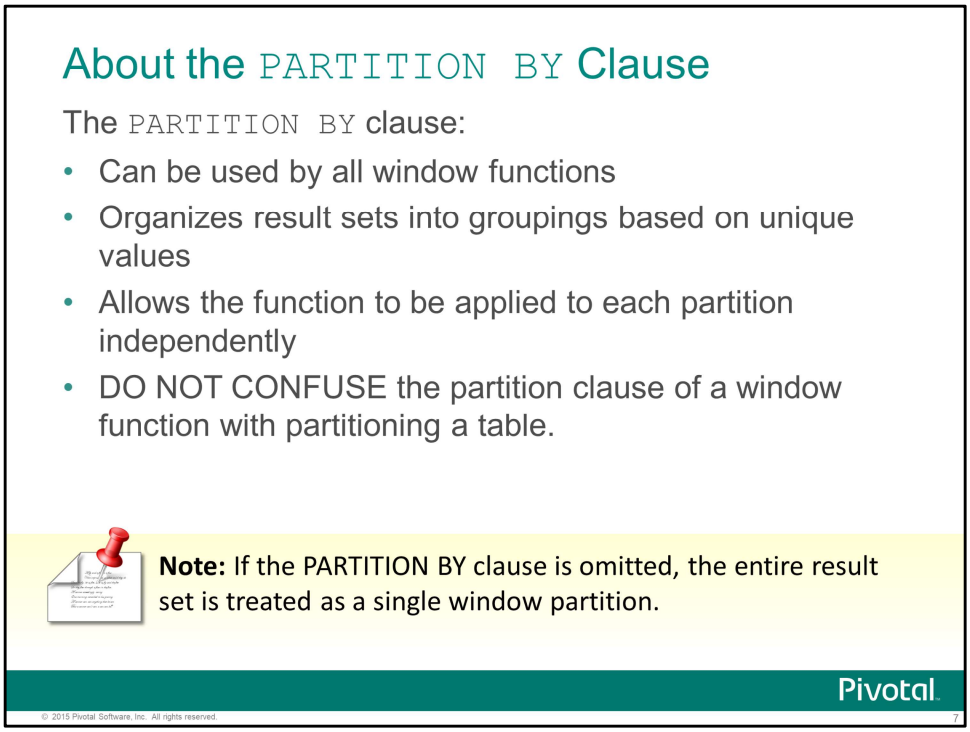
- Partitions (groupings) – a window function calculates the results for a row in respect to its partition.
- Ordering of rows within a window partition – some window function such as RANK require ordering.
- Framing – for ordered result sets, you can define a window frame that analyzes each row with respect to the rows directly above or below it.




All window functions must have an `OVER()` clause. The window function specifies the window of data to which the function applies.

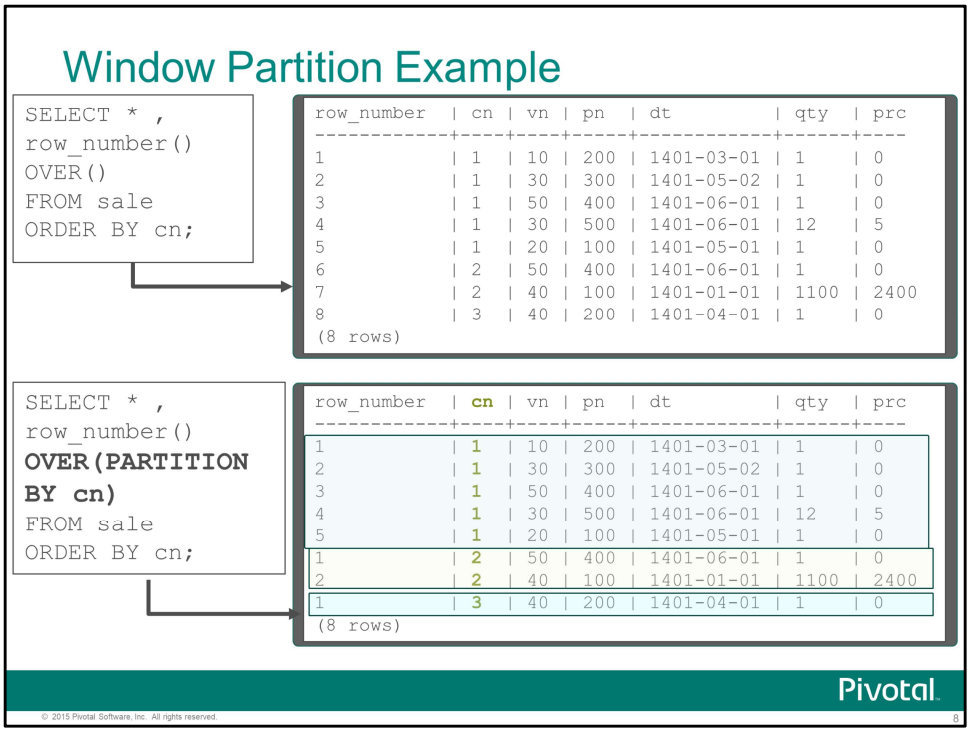
It defines:

- Window partitions using the `PARTITION BY` clause.
- Ordering within a window partition using the `ORDER BY` clause).
- Framing within a window partition (`ROWS/RANGE` clauses).




The `PARTITION BY` clause:

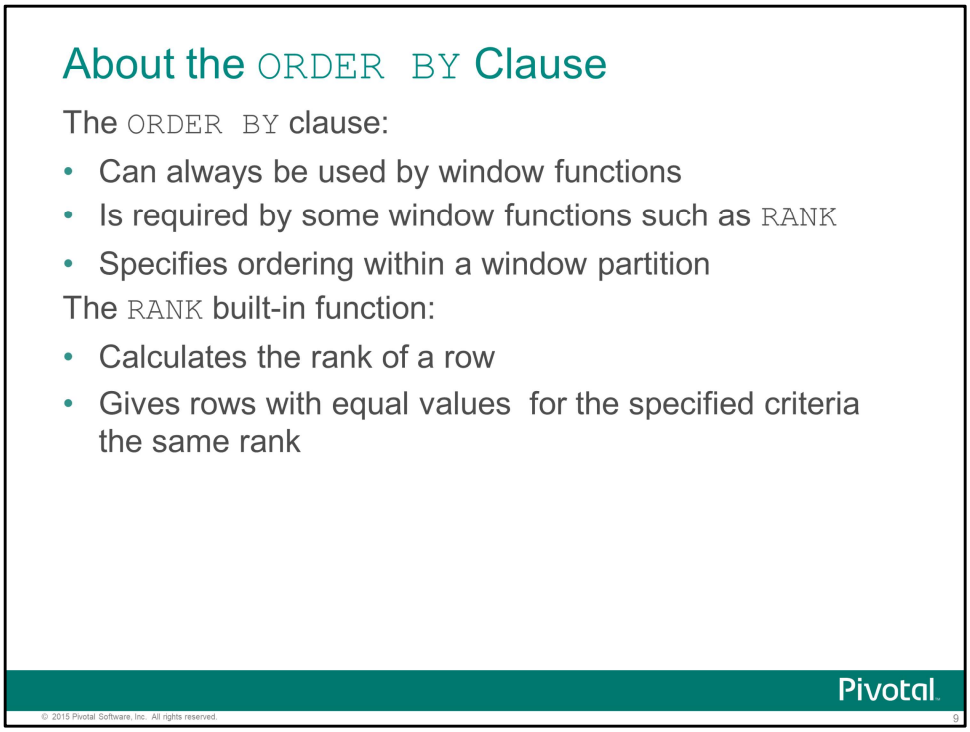
- Can be used by all window functions. However, it is not a required clause. Windows that do not use the `PARTITION BY` clause present the entire result set as a single window partition.
- Organizes the result set into groupings based on the unique values of the specified expression or column.
- Allows the function to be applied to each partition independently.




The example on the slide uses the `row_number` window function. This function returns a row number for each unique row in the result set.

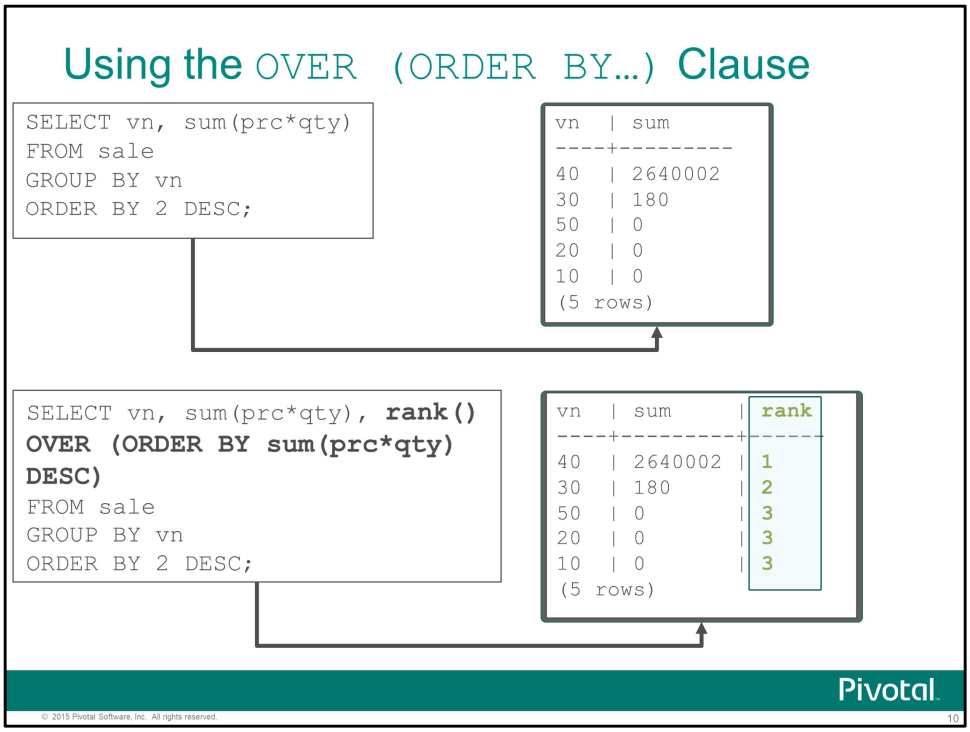
In the first example, the `OVER` clause does not have a `PARTITION BY`. The entire result set is treated as one window partition.

In the second example, the window is partitioned by the customer number. Note that the result of row number is calculated within each window partition.




The `ORDER BY` clause is used to order the resulting data set based on an expression or column. It is always allowed in windows functions and is required by some window functions, including `RANK`. The `ORDER BY` clause specifies ordering within a window partition.

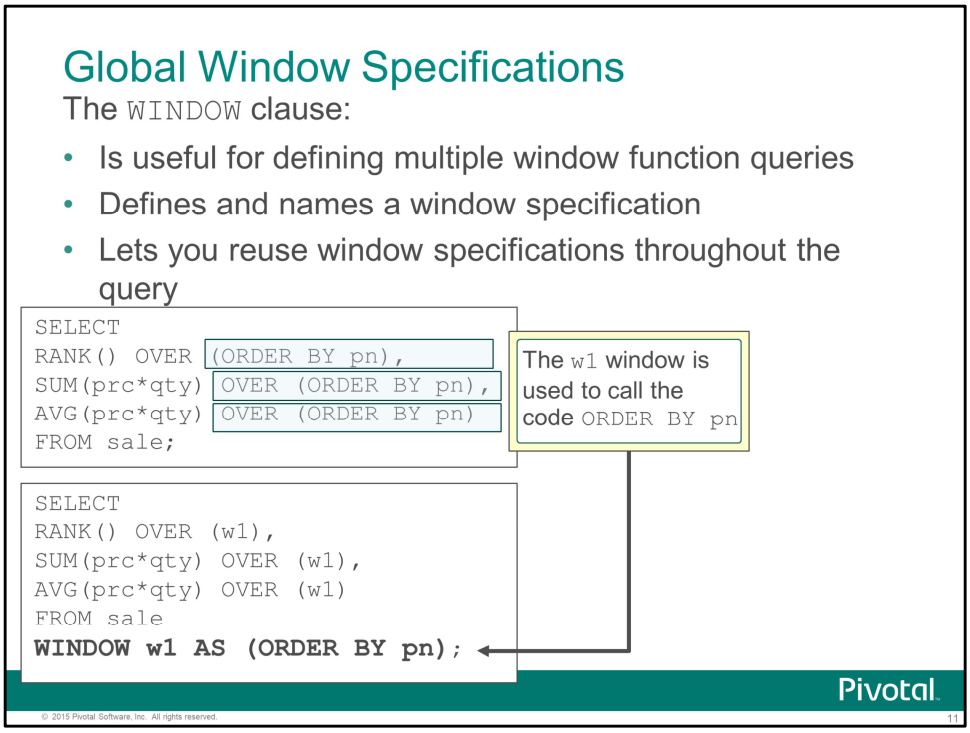
The RANK function is a built-in function that calculates the rank of a row in an ordered group of values. Rows with equal values for the ranking criteria receive the same rank. The number of tied rows are added to the rank number to calculate the next rank value. In this case, ranks may not be consecutive numbers.




The slide shows an example of two queries that rank vendors by sales totals.

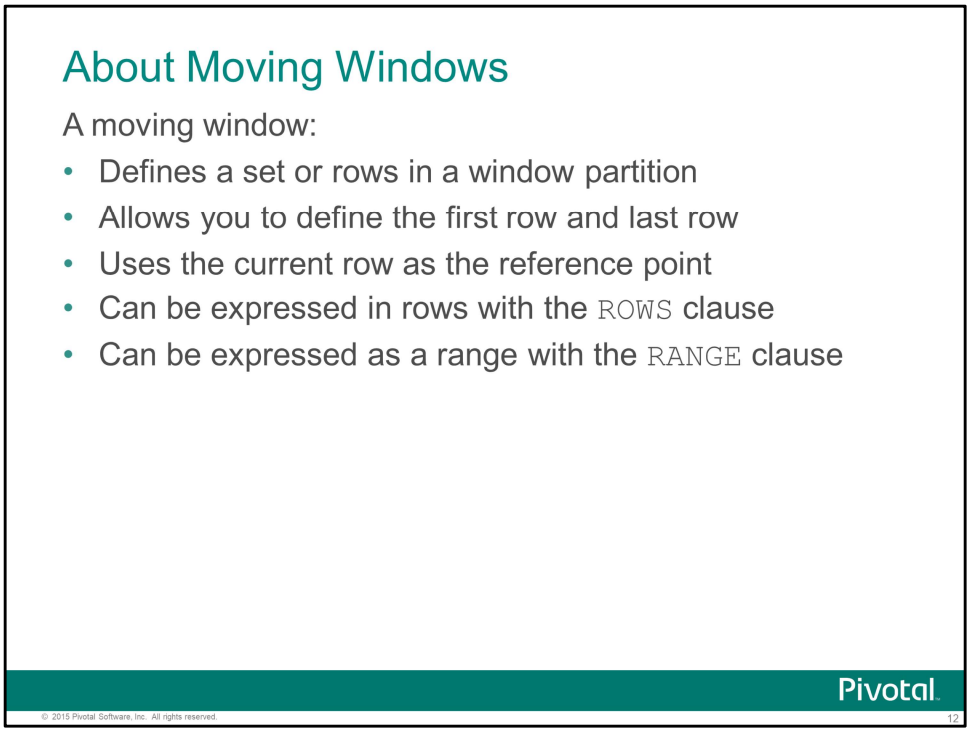
The first query shows a window function grouped on the vendor column, `vn`.

The second query uses the `RANK` function to output a ranking number for each row. Note that the `PARTITION BY` clause is not used in this query. The entire result is one window partition. Also, do not confuse `ORDER BY` of a window specification with the `ORDER BY` of a query.




The `WINDOW` clause is a convenient feature that allows you to define and name a window specification once and then refer to it multiple times throughout the query.

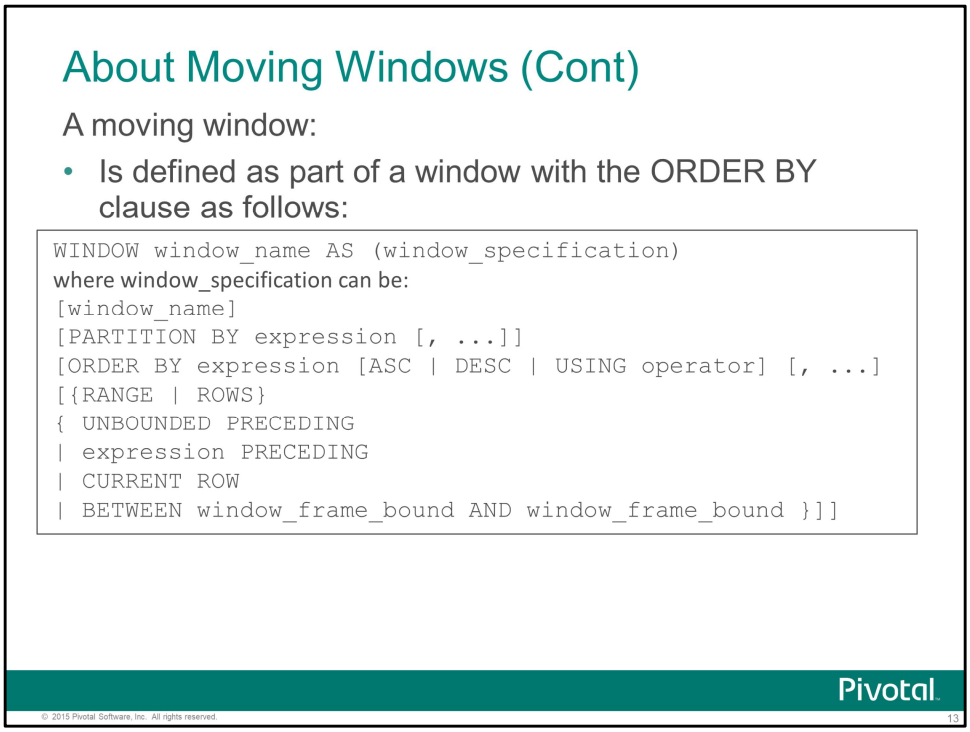
In the example shown on the slide, several window functions are using the same window specification in the `OVER` clause. You can use the `WINDOW` clause to define the window specification once, and then refer to it by name. In this example, the window specification is called `w1`.



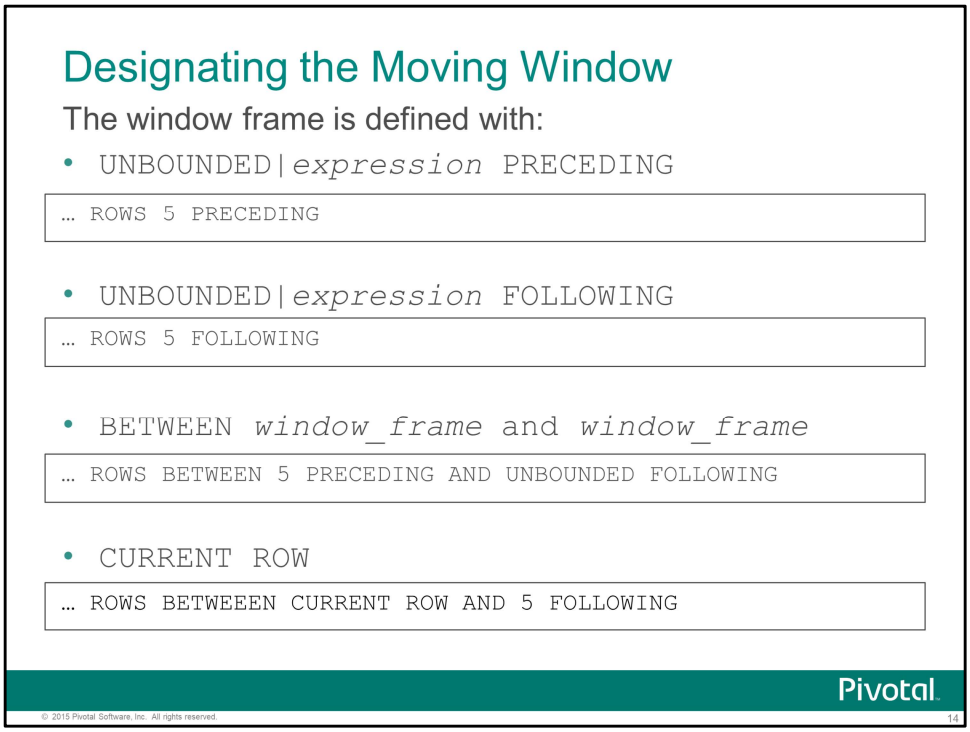

A moving or rolling window defines a set of rows within a window partition. When you define a window frame, the window function is computed with respect to the contents of this moving frame, rather than against the fixed contents of the entire window partition. Window frames can be row-based, represented by the `ROWS` clause, or value based, represented by a `RANGE`.

When the window frame is row-based, you define the number of rows offset from the current row. If the window frame is range-based, you define the bounds of the window frame in terms of data values offset from the value in the current row.

If you specify only a starting row for the window, the current row is used as the last row in the window.




About Moving Windows (Continued)




The window frame can be defined as:

- **`UNBOUNDED` or *`expression`* `PRECEDING`** – This clause defines the first row of the window using the current row as a reference point. The starting row is expressed in terms of the number of rows preceding the current row. If you define the window frame for a `ROWS` window frame as `5 PRECEDING`, the window frame starts at the fifth row preceding the current row. If the definition is for a RANGE window frame, the window starts with the first row whose ordering column value precedes that of the current row by 5. If the term `UNBOUNDED` is used, the first row of the partition acts as the first row of the window.
- **`UNBOUNDED` or *`expression`* `FOLLOWING`** – This clause defines the last rows of the window using the current row as a reference point. Similar to `PRECEDING`, the last row is expressed in terms of the number or rows following the current row. Either an expression or the term `UNBOUNDED` can be used to identify the last rows. If `UNBOUNDED` is used, the last row in the window is the last row in the partition.

## Built-in Window Functions

| Built-in Function | Description |
|---|---|
| `cume_dist()` | Calculates the cumulative distribution of a value in a group of values. Rows with equal values always evaluate to the same cumulative distribution value. |
| `dense_rank()` | Computes the rank of a row in an ordered group of rows without skipping rank values. Rows with equal values are given the same rank value. |
| `first_value(`*`expr`*`)` | Returns the first value in an ordered set of values. |
| `lag(expr [,`*`offset`*`] [,default])` | Provides access to more than one row of the same table without doing a self join. Given a series of rows returned from a query and a position of the cursor, LAG provides access to a row at a given physical offset prior to that position. If offset is not specified, the default offset is 1. default sets the value that is returned if the offset goes beyond the scope of the window. If default is not specified, the default value is null. |

**Note:** Any aggregate function used with the `OVER` clause can also be used as a window function.

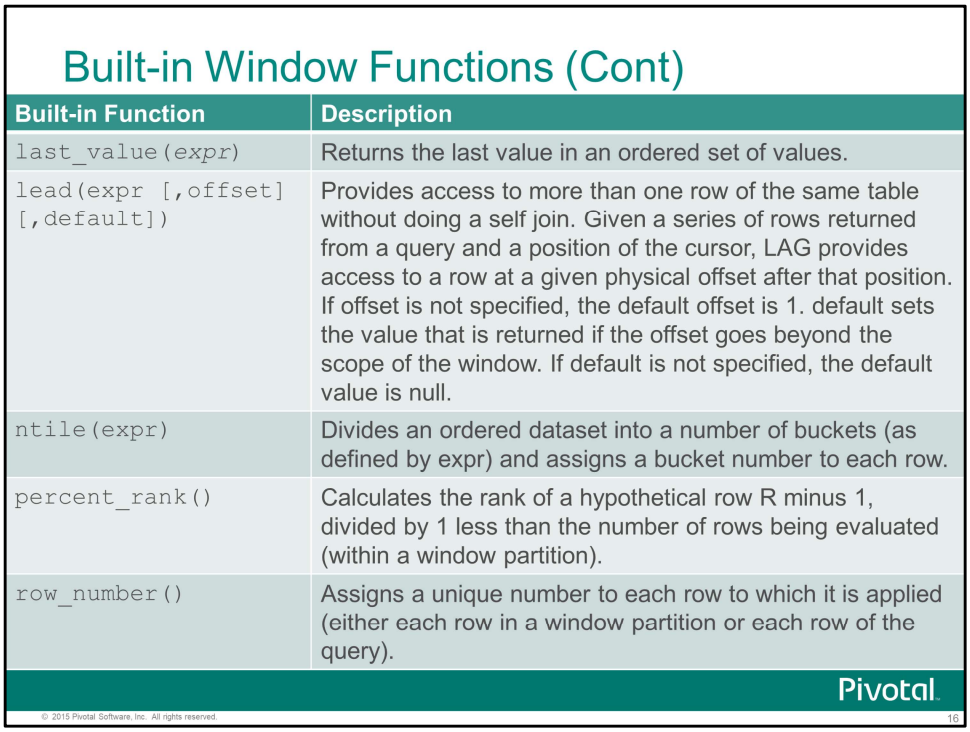
Pivotal





The slide shows built-in window functions supported within Greenplum. These built-in functions require an `OVER` clause.

For more detailed information on the functions, refer to the *Greenplum Database Administrator Guide*.




Built-in Window Functions (Continued)

# Real Example Problem

United Widget has a production process that makes widgets in lots and runs tests on them during the process. Testing devices produce a tuple with the following information:

- Lot ID (unique globally)
- Widget ID (unique within lot, not globally)
- Test Time (timestamp)
- Defect ID (a non-negative integer – 0 is a special value)
- Defect Count (# of defects of this type found at lot, widget, defect, time)

Widget production analysts want to calculate the following statistics:

- For each (lot, widget, test time, defect) the percentage of non-zero defects
- For each (lot, widget, test time, defect) the total number of defects, including type 0

Pivotal

# Example Data

| Lot ID | Widget ID | Test time | Defect ID | Defect Count |
|---|---|---|---|---|
| 1 | 1 | 5/1/2015 12:43 | 0 | 3 |
| 1 | 1 | 5/1/2015 12:43 | 1 | 2 |
| 1 | 1 | 5/1/2015 13:04 | 0 | 5 |
| 1 | 1 | 5/1/2015 13:04 | 2 | 4 |
| 1 | 2 | 5/1/2015 12:43 | 0 | 2 |
| 1 | 2 | 5/1/2015 12:43 | 1 | 3 |
| 1 | 2 | 5/1/2015 12:43 | 2 | 4 |
| 1 | 2 | 5/1/2015 12:43 | 3 | 1 |
| 1 | 2 | 5/1/2015 13:04 | 1 | 1 |
| 1 | 2 | 5/1/2015 13:04 | 2 | 3 |
| 1 | 2 | 5/1/2015 13:04 | 3 | 11 |
| 1 | 2 | 5/1/2015 13:04 | 4 | 4 |
| 1 | 2 | 5/1/2015 13:04 | 5 | 6 |

Pivotal

# Desired Result

| Lot ID | Widget ID | Test Time | Defect ID | defects | Pct_def | Total Defects |
|---|---|---|---|---|---|---|
| 1 | 1 | 5/1/2015 12:43 | 0 | 3 | - | (3+2) |
| 1 | 1 | 5/1/2015 12:43 | 1 | 2 | 2/2 | (3+2) |
| 1 | 1 | 5/1/2015 13:04 | 0 | 5 | - | (5+4) |
| 1 | 1 | 5/1/2015 13:04 | 2 | 4 | 4/4 | (5+4) |
| 1 | 2 | 5/1/2015 12:43 | 0 | 2 | - | (1+2+3+4) |
| 1 | 2 | 5/1/2015 12:43 | 1 | 3 | 3/(3+1+4) | (1+2+3+4) |
| 1 | 2 | 5/1/2015 12:43 | 2 | 4 | 4/(3+1+4) | (1+2+3+4) |
| 1 | 2 | 5/1/2015 12:43 | 3 | 1 | 1(3+1+4) | (1+2+3+4) |
| 1 | 2 | 5/1/2015 13:04 | 1 | 1 | 1/(1+3+11+4+6) | (1+3+11+4+6) |
| 1 | 2 | 5/1/2015 13:04 | 2 | 3 | 3/(1+3+11+4+6) | (1+3+11+4+6) |
| 1 | 2 | 5/1/2015 13:04 | 3 | 11 | 11/(1+3+11+4+6) | (1+3+11+4+6) |
| 1 | 2 | 5/1/2015 13:04 | 4 | 4 | 4/(1+3+11+4+6) | (1+3+11+4+6) |
| 1 | 2 | 5/1/2015 13:04 | 5 | 6 | 6/(1+3+11+4+6) | (1+3+11+4+6) |

Pivotal

## Additional resources

- http://blog.pivotal.io/data-science-pivotal/products/time-series-analysis-1-introduction-to-window-functions
- http://blog.pivotal.io/data-science-pivotal/products/time-series-analysis-2-recognizing-patterns-within-a-time-series
- http://blog.pivotal.io/data-science-pivotal/products/time-series-analysis-part-3-resampling-and-interpolation
- http://www.postgresql.org/docs/9.1/static/tutorial-window.html
- http://tapoueh.org/blog/2013/08/20-Window-Functions
- http://sqlschool.modeanalytics.com/advanced/window-functions.html
- https://www.pgcon.org/2009/schedule/attachments/98_Windowing%20Functions.pdf

Pivotal

## Agenda

- Window Functions
  - What are they?
  - How do recognize one?
  - How are they written?
  - Some examples
- References, Review

Pivotal

# Thank You

Pivotal


Pivotal
A NEW PLATFORM FOR A NEW ERA