Window functions

```
import spark.implicits._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
```

```
import spark.implicits._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
```

```
case class Salary(depName: String, empNo: Long, salary: Long)
```

```
defined class Salary
```

```
val empsalary = Seq(
  Salary("sales", 1, 5000),
  Salary("personnel", 2, 3900),
  Salary("sales", 3, 4800),
  Salary("sales", 4, 4800),
  Salary("sales", 6, 4500),
  Salary("sales", 12, 5000),
  Salary("personnel", 5, 3500),
  Salary("develop", 7, 4200),
  Salary("develop", 8, 6000),
  Salary("develop", 9, 4500),
  Salary("develop", 10, 5200),
  Salary("develop", 11, 5200)).toDS
```

```
empsalary: org.apache.spark.sql.Dataset[Salary] = [depName: string, empNo: bigint ... 1 more field]
```

```
empsalary.show
```

```
+---------+-----+------+
|  depName|empNo|salary|
+---------+-----+------+
|    sales|    1|  5000|
|personnel|    2|  3900|
|    sales|    3|  4800|
|    sales|    4|  4800|
|    sales|    6|  4500|
|    sales|   12|  5000|
|personnel|    5|  3500|
|  develop|    7|  4200|
|  develop|    8|  6000|
|  develop|    9|  4500|
|  develop|   10|  5200|
|  develop|   11|  5200|
+---------+-----+------+
```

```
val byDepName = Window.partitionBy($"depName")
```

```
byDepName: org.apache.spark.sql.expressions.WindowSpec = org.apache.spark.sql.expressions.WindowSpec@75d89b5d
```

```
empsalary.withColumn("avg", avg($"salary").over(byDepName)).show
```

```
+---------+-----+------+------+
|  depName|empNo|salary|   avg|
+---------+-----+------+------+
|  develop|    7|  4200|5020.0|
|  develop|    8|  6000|5020.0|
|  develop|    9|  4500|5020.0|
|  develop|   10|  5200|5020.0|
|  develop|   11|  5200|5020.0|
|    sales|    1|  5000|4820.0|
|    sales|    3|  4800|4820.0|
|    sales|    4|  4800|4820.0|
|    sales|    6|  4500|4820.0|
|    sales|   12|  5000|4820.0|
|personnel|    2|  3900|3700.0|
|personnel|    5|  3500|3700.0|
+---------+-----+------+------+
```

```
empsalary.withColumn("avg", avg('salary) over byDepName).show
```

```
+---------+-----+------+------+
|  depName|empNo|salary|   avg|
+---------+-----+------+------+
|  develop|    7|  4200|5020.0|
|  develop|    8|  6000|5020.0|
|  develop|    9|  4500|5020.0|
|  develop|   10|  5200|5020.0|
|  develop|   11|  5200|5020.0|
|    sales|    1|  5000|4820.0|
|    sales|    3|  4800|4820.0|
|    sales|    4|  4800|4820.0|
|    sales|    6|  4500|4820.0|
|    sales|   12|  5000|4820.0|
|personnel|    2|  3900|3700.0|
|personnel|    5|  3500|3700.0|
+---------+-----+------+------+
```

```
empsalary.select('depName, 'empNo, avg('salary).over(byDepName).as("avg")).show
```

```
+---------+-----+------+
|  depName|empNo|   avg|
+---------+-----+------+
|  develop|    7|5020.0|
|  develop|    8|5020.0|
|  develop|    9|5020.0|
|  develop|   10|5020.0|
```

```
|  develop|   11|5020.0|
|    sales|    1|4820.0|
|    sales|    3|4820.0|
|    sales|    4|4820.0|
|    sales|    6|4820.0|
|    sales|   12|4820.0|
|personnel|    2|3700.0|
|personnel|    5|3700.0|
+---------+-----+------+
```

```
// Orderby on the DF not window group
empsalary.withColumn("avg", avg('salary).over(byDepName)).orderBy('salary).show
```

```
+---------+-----+------+------+
|  depName|empNo|salary|   avg|
+---------+-----+------+------+
|personnel|    5|  3500|3700.0|
|personnel|    2|  3900|3700.0|
|  develop|    7|  4200|5020.0|
|  develop|    9|  4500|5020.0|
|    sales|    6|  4500|4820.0|
|    sales|    4|  4800|4820.0|
|    sales|    3|  4800|4820.0|
|    sales|    1|  5000|4820.0|
|    sales|   12|  5000|4820.0|
|  develop|   10|  5200|5020.0|
|  develop|   11|  5200|5020.0|
|  develop|    8|  6000|5020.0|
+---------+-----+------+------+
```

```
// Ordering in window
```

```
val byDepNameSalaryDesc = Window.partitionBy('depName).orderBy('salary.desc)
```

```
byDepNameSalaryDesc: org.apache.spark.sql.expressions.WindowSpec = org.apache.spark.sql.expressions.WindowSpec@7da730b0
```

```
empsalary.withColumn("avg", avg('salary).over(byDepNameSalaryDesc)).show
```

```
+---------+-----+------+-----------------+
|  depName|empNo|salary|              avg|
+---------+-----+------+-----------------+
|  develop|    8|  6000|           6000.0|
|  develop|   10|  5200|5466.666666666667|
|  develop|   11|  5200|5466.666666666667|
|  develop|    9|  4500|           5225.0|
|  develop|    7|  4200|           5020.0|
|    sales|    1|  5000|           5000.0|
|    sales|   12|  5000|           5000.0|
|    sales|    3|  4800|           4900.0|
|    sales|    4|  4800|           4900.0|
|    sales|    6|  4500|           4820.0|
|personnel|    2|  3900|           3900.0|
|personnel|    5|  3500|           3700.0|
+---------+-----+------+-----------------+
```

```
// Rank (NOTE how the rank values are skipped)
```

```
empsalary.withColumn("rank", rank.over(byDepNameSalaryDesc)).show
```

```
+---------+-----+------+----+
|  depName|empNo|salary|rank|
+---------+-----+------+----+
|  develop|    8|  6000|   1|
|  develop|   10|  5200|   2|
|  develop|   11|  5200|   2|
|  develop|    9|  4500|   4|
|  develop|    7|  4200|   5|
|    sales|    1|  5000|   1|
|    sales|   12|  5000|   1|
|    sales|    3|  4800|   3|
|    sales|    4|  4800|   3|
|    sales|    6|  4500|   5|
|personnel|    2|  3900|   1|
|personnel|    5|  3500|   2|
+---------+-----+------+----+
```

```
// With dense_rank, rank values are not skipped
```

```
empsalary.withColumn("rank", dense_rank.over(byDepNameSalaryDesc)).show
```

```
+---------+-----+------+----+
|  depName|empNo|salary|rank|
+---------+-----+------+----+
|  develop|    8|  6000|   1|
|  develop|   10|  5200|   2|
|  develop|   11|  5200|   2|
|  develop|    9|  4500|   3|
|  develop|    7|  4200|   4|
|    sales|    1|  5000|   1|
|    sales|   12|  5000|   1|
|    sales|    3|  4800|   2|
|    sales|    4|  4800|   2|
|    sales|    6|  4500|   3|
|personnel|    2|  3900|   1|
|personnel|    5|  3500|   2|
+---------+-----+------+----+
```

Frame

```
// Max of salary, frame start = one row prior to current row, frame end = current row

empsalary.withColumn("max", max('salary).over(
    Window.partitionBy('depName).orderBy('salary).rowsBetween(-1, Window.currentRow))).show

+---------+-----+------+----+
|  depName|empNo|salary| max|
+---------+-----+------+----+
|  develop|    7|  4200|4200|
|  develop|    9|  4500|4500|
|  develop|   10|  5200|5200|
|  develop|   11|  5200|5200|
|  develop|    8|  6000|6000|
|    sales|    6|  4500|4500|
|    sales|    3|  4800|4800|
|    sales|    4|  4800|4800|
|    sales|    1|  5000|5000|
|    sales|   12|  5000|5000|
|personnel|    5|  3500|3500|
|personnel|    2|  3900|3900|
+---------+-----+------+----+

// Max of salary, frame start = one row prior to current row, frame end = one row following current row

empsalary.withColumn("max", max('salary).over(
    Window.partitionBy('depName).orderBy('salary).rowsBetween(-1, 1))).show

+---------+-----+------+----+
|  depName|empNo|salary| max|
+---------+-----+------+----+
|  develop|    7|  4200|4500|
|  develop|    9|  4500|5200|
|  develop|   10|  5200|5200|
|  develop|   11|  5200|6000|
|  develop|    8|  6000|6000|
|    sales|    6|  4500|4800|
|    sales|    3|  4800|4800|
|    sales|    4|  4800|5000|
|    sales|    1|  5000|5000|
|    sales|   12|  5000|5000|
|personnel|    5|  3500|3900|
|personnel|    2|  3900|3900|
+---------+-----+------+----+

// Max of salary, frame start = all rows prior to current row, frame end = current row

empsalary.withColumn("max", max('salary).over(
    Window.partitionBy('depName).orderBy('salary).rowsBetween(Window.unboundedPreceding, Window.currentRow))).show

+---------+-----+------+----+
|  depName|empNo|salary| max|
+---------+-----+------+----+
|  develop|    7|  4200|4200|
|  develop|    9|  4500|4500|
|  develop|   10|  5200|5200|
|  develop|   11|  5200|5200|
|  develop|    8|  6000|6000|
|    sales|    6|  4500|4500|
|    sales|    3|  4800|4800|
|    sales|    4|  4800|4800|
|    sales|    1|  5000|5000|
|    sales|   12|  5000|5000|
|personnel|    5|  3500|3500|
|personnel|    2|  3900|3900|
+---------+-----+------+----+

// Max of salary, frame start = current row, frame end = all rows following current row

empsalary.withColumn("max", max('salary).over(
    Window.partitionBy('depName).orderBy('salary).rowsBetween(Window.currentRow, Window.unboundedFollowing))).show

+---------+-----+------+----+
|  depName|empNo|salary| max|
+---------+-----+------+----+
|  develop|    7|  4200|6000|
|  develop|    9|  4500|6000|
|  develop|   10|  5200|6000|
|  develop|   11|  5200|6000|
|  develop|    8|  6000|6000|
|    sales|    6|  4500|5000|
|    sales|    3|  4800|5000|
|    sales|    4|  4800|5000|
|    sales|    1|  5000|5000|
|    sales|   12|  5000|5000|
|personnel|    5|  3500|3900|
|personnel|    2|  3900|3900|
+---------+-----+------+----+

// Max of salary, frame start = all rows prior to current row, frame end = all rows following current row

empsalary.withColumn("max", max('salary).over(
    Window.partitionBy('depName).orderBy('salary).rowsBetween(Window.unboundedPreceding, Window.unboundedFollowing))).show

+---------+-----+------+----+
|  depName|empNo|salary| max|
+---------+-----+------+----+
|  develop|    7|  4200|6000|
|  develop|    9|  4500|6000|
|  develop|   10|  5200|6000|
|  develop|   11|  5200|6000|
```

```
|  develop|    8|  6000|6000|
|    sales|    6|  4500|5000|
|    sales|    3|  4800|5000|
|    sales|    4|  4800|5000|
|    sales|    1|  5000|5000|
|    sales|   12|  5000|5000|
|personnel|    5|  3500|3900|
|personnel|    2|  3900|3900|
+---------+-----+------+----+
```

```
// Same as above

// Max of salary, frame start = one row prior to current row, frame end = one row following current row

empsalary.withColumn("max", max('salary).over(
    Window.partitionBy('depName).orderBy('salary.desc))).show
```

```
+---------+-----+------+----+
|  depName|empNo|salary| max|
+---------+-----+------+----+
|  develop|    8|  6000|6000|
|  develop|   10|  5200|6000|
|  develop|   11|  5200|6000|
|  develop|    9|  4500|6000|
|  develop|    7|  4200|6000|
|    sales|    1|  5000|5000|
|    sales|   12|  5000|5000|
|    sales|    3|  4800|5000|
|    sales|    4|  4800|5000|
|    sales|    6|  4500|5000|
|personnel|    2|  3900|3900|
|personnel|    5|  3500|3900|
+---------+-----+------+----+
```

Note: I don't understand the below behavior

Update

- Frame or Window is determined by the field mentioned in orderBy()
- Result (max in the below case) is determined by comparing the current row with the previous row
- So, if you want to find max of salary withing a window order salary in desc order
- if you want to find min of salary withing a window order salary in asc order

```
// NOTE: I don't understand this

empsalary.withColumn("max", max('salary).over(
    Window.partitionBy('depName).orderBy('salary.asc))).show
```

```
+---------+-----+------+----+
|  depName|empNo|salary| max|
+---------+-----+------+----+
|  develop|    7|  4200|4200|
|  develop|    9|  4500|4500|
|  develop|   10|  5200|5200|
|  develop|   11|  5200|5200|
|  develop|    8|  6000|6000|
|    sales|    6|  4500|4500|
|    sales|    3|  4800|4800|
|    sales|    4|  4800|4800|
|    sales|    1|  5000|5000|
|    sales|   12|  5000|5000|
|personnel|    5|  3500|3500|
|personnel|    2|  3900|3900|
+---------+-----+------+----+
```

Looks like it compares the previous value to find the current result

```
empsalary.withColumn("max", max('salary).over(
    Window.partitionBy('depName).orderBy('empNo))).show
```

```
+---------+-----+------+----+
|  depName|empNo|salary| max|
+---------+-----+------+----+
|  develop|    7|  4200|4200|
|  develop|    8|  6000|6000|
|  develop|    9|  4500|6000|
|  develop|   10|  5200|6000|
|  develop|   11|  5200|6000|
|    sales|    1|  5000|5000|
|    sales|    3|  4800|5000|
|    sales|    4|  4800|5000|
|    sales|    6|  4500|5000|
|    sales|   12|  5000|5000|
|personnel|    2|  3900|3900|
|personnel|    5|  3500|3900|
+---------+-----+------+----+
```

frame is determined by the field mentioned in orderBy()

```
empsalary.withColumn("max", max('salary).over(
    Window.partitionBy('depName).orderBy('empNo).rangeBetween(-1, Window.currentRow))).show
```

```
+---------+-----+------+----+
|  depName|empNo|salary| max|
+---------+-----+------+----+
|  develop|    7|  4200|4200|
```

```
|  develop|    8|  6000|6000|
|  develop|    9|  4500|6000|
|  develop|   10|  5200|5200|
|  develop|   11|  5200|5200|
|    sales|    1|  5000|5000|
|    sales|    3|  4800|4800|
|    sales|    4|  4800|4800|
|    sales|    6|  4500|4500|
|    sales|   12|  5000|5000|
|personnel|    2|  3900|3900|
|personnel|    5|  3500|3500|
+---------+-----+------+----+
```

```
val dataset = Seq(
  ("Thin",       "cell phone", 6000),
  ("Normal",     "tablet",     1500),
  ("Mini",       "tablet",     5500),
  ("Ultra thin", "cell phone", 5000),
  ("Very thin",  "cell phone", 6000),
  ("Big",        "tablet",     2500),
  ("Bendable",   "cell phone", 3000),
  ("Foldable",   "cell phone", 3000),
  ("Pro",        "tablet",     4500),
  ("Pro2",       "tablet",     6500))
  .toDF("product", "category", "revenue")
```

```
dataset: org.apache.spark.sql.DataFrame = [product: string, category: string ... 1 more field]
```

```
dataset.show
```

```
+----------+----------+-------+
|   product|  category|revenue|
+----------+----------+-------+
|      Thin|cell phone|   6000|
|    Normal|    tablet|   1500|
|      Mini|    tablet|   5500|
|Ultra thin|cell phone|   5000|
| Very thin|cell phone|   6000|
|       Big|    tablet|   2500|
|  Bendable|cell phone|   3000|
|  Foldable|cell phone|   3000|
|       Pro|    tablet|   4500|
|      Pro2|    tablet|   6500|
+----------+----------+-------+
```

Find the best selling and 2nd best selling product in each category

```
val byCategoryRevenueDesc = Window.partitionBy('category).orderBy('revenue.desc)
```

```
byCategoryRevenueDesc: org.apache.spark.sql.expressions.WindowSpec = org.apache.spark.sql.expressions.WindowSpec@4462f528
```

```
val ranked = dataset.withColumn("dense_rank", dense_rank.over(byCategoryRevenueDesc))
```

```
ranked: org.apache.spark.sql.DataFrame = [product: string, category: string ... 2 more fields]
```

```
ranked.show
```

```
+----------+----------+-------+----------+
|   product|  category|revenue|dense_rank|
+----------+----------+-------+----------+
|      Pro2|    tablet|   6500|         1|
|      Mini|    tablet|   5500|         2|
|       Pro|    tablet|   4500|         3|
|       Big|    tablet|   2500|         4|
|    Normal|    tablet|   1500|         5|
|      Thin|cell phone|   6000|         1|
| Very thin|cell phone|   6000|         1|
|Ultra thin|cell phone|   5000|         2|
|  Bendable|cell phone|   3000|         3|
|  Foldable|cell phone|   3000|         3|
+----------+----------+-------+----------+
```

```
dataset.withColumn("percent_rank", percent_rank.over(byCategoryRevenueDesc)).show
```

```
+----------+----------+-------+------------+
|   product|  category|revenue|percent_rank|
+----------+----------+-------+------------+
|      Pro2|    tablet|   6500|         0.0|
|      Mini|    tablet|   5500|        0.25|
|       Pro|    tablet|   4500|         0.5|
|       Big|    tablet|   2500|        0.75|
|    Normal|    tablet|   1500|         1.0|
|      Thin|cell phone|   6000|         0.0|
| Very thin|cell phone|   6000|         0.0|
|Ultra thin|cell phone|   5000|         0.5|
|  Bendable|cell phone|   3000|        0.75|
|  Foldable|cell phone|   3000|        0.75|
+----------+----------+-------+------------+
```

```
ranked.where('rank <= 2).show
```

```
+----------+----------+-------+----+
|   product|  category|revenue|rank|
+----------+----------+-------+----+
|      Pro2|    tablet|   6500|   1|
|      Mini|    tablet|   5500|   2|
|      Thin|cell phone|   6000|   1|
| Very thin|cell phone|   6000|   1|
|Ultra thin|cell phone|   5000|   2|
+----------+----------+-------+----+
```

Revenue difference per category

```
dataset.withColumn("max", max('revenue).over(byCategoryRevenueDesc)).show
```

```
+----------+----------+-------+----+
|   product|  category|revenue| max|
+----------+----------+-------+----+
|      Pro2|    tablet|   6500|6500|
|      Mini|    tablet|   5500|6500|
|       Pro|    tablet|   4500|6500|
|       Big|    tablet|   2500|6500|
|    Normal|    tablet|   1500|6500|
|      Thin|cell phone|   6000|6000|
| Very thin|cell phone|   6000|6000|
|Ultra thin|cell phone|   5000|6000|
|  Bendable|cell phone|   3000|6000|
|  Foldable|cell phone|   3000|6000|
+----------+----------+-------+----+
```

```
val diffColumn = max('revenue).over(byCategoryRevenueDesc) - 'revenue
```

```
diffColumn: org.apache.spark.sql.Column = (max(revenue) OVER (PARTITION BY category ORDER BY revenue DESC NULLS LAST unspecifiedframe$()) - revenue)
```

```
dataset.select($"*", diffColumn.as("revenue_diff")).show
```

```
+----------+----------+-------+------------+
|   product|  category|revenue|revenue_diff|
+----------+----------+-------+------------+
|      Pro2|    tablet|   6500|           0|
|      Mini|    tablet|   5500|        1000|
|       Pro|    tablet|   4500|        2000|
|       Big|    tablet|   2500|        4000|
|    Normal|    tablet|   1500|        5000|
|      Thin|cell phone|   6000|           0|
| Very thin|cell phone|   6000|           0|
|Ultra thin|cell phone|   5000|        1000|
|  Bendable|cell phone|   3000|        3000|
|  Foldable|cell phone|   3000|        3000|
+----------+----------+-------+------------+
```

```
dataset.withColumn("revenue_diff", diffColumn).show
```

```
+----------+----------+-------+------------+
|   product|  category|revenue|revenue_diff|
+----------+----------+-------+------------+
|      Pro2|    tablet|   6500|           0|
|      Mini|    tablet|   5500|        1000|
|       Pro|    tablet|   4500|        2000|
|       Big|    tablet|   2500|        4000|
|    Normal|    tablet|   1500|        5000|
|      Thin|cell phone|   6000|           0|
| Very thin|cell phone|   6000|           0|
|Ultra thin|cell phone|   5000|        1000|
|  Bendable|cell phone|   3000|        3000|
|  Foldable|cell phone|   3000|        3000|
+----------+----------+-------+------------+
```

lead

```
val pairs = for {
    x <- 1 to 5
    y <- 1 to 2
} yield (x, 10 * x * y)

val ds = pairs.toDF("ns", "tens")
ds.show
```

```
+---+----+
| ns|tens|
+---+----+
|  1|  10|
|  1|  20|
|  2|  20|
|  2|  40|
|  3|  30|
|  3|  60|
|  4|  40|
|  4|  80|
|  5|  50|
|  5| 100|
+---+----+
```

```
pairs: scala.collection.immutable.IndexedSeq[(Int, Int)] = Vector((1,10), (1,20), (2,20), (2,40), (3,30), (3,60), (4,40), (4,80), (5,50), (5,100))
ds: org.apache.spark.sql.DataFrame = [ns: int, tens: int]
```

```
val overNs = Window.partitionBy('ns).orderBy('tens)
ds.withColumn("lead", lead('tens, 1).over(overNs)).show
```

```
+---+----+----+
| ns|tens|lead|
+---+----+----+
|  1|  10|  20|
|  1|  20|null|
|  3|  30|  60|
|  3|  60|null|
|  5|  50| 100|
|  5| 100|null|
|  4|  40|  80|
|  4|  80|null|
```

```
|  2|  20|  40|
|  2|  40|null|
+---+----+----+

overNs: org.apache.spark.sql.expressions.WindowSpec = org.apache.spark.sql.expressions.WindowSpec@5f5b22e9
```

Running total

```
val sales = Seq(
  (0, 0, 0, 5),
  (1, 0, 1, 3),
  (2, 0, 2, 1),
  (3, 1, 0, 2),
  (4, 2, 0, 8),
  (5, 2, 2, 8))
  .toDF("id", "orderID", "prodID", "orderQty")

sales: org.apache.spark.sql.DataFrame = [id: int, orderID: int ... 2 more fields]

sales.show

+---+-------+------+--------+
| id|orderID|prodID|orderQty|
+---+-------+------+--------+
|  0|      0|     0|       5|
|  1|      0|     1|       3|
|  2|      0|     2|       1|
|  3|      1|     0|       2|
|  4|      2|     0|       8|
|  5|      2|     2|       8|
+---+-------+------+--------+

val byId = Window.orderBy('id)

byId: org.apache.spark.sql.expressions.WindowSpec = org.apache.spark.sql.expressions.WindowSpec@3bc13d2f

sales.withColumn("running_total", sum('orderQty).over(byId)).show

+---+-------+------+--------+-------------+
| id|orderID|prodID|orderQty|running_total|
+---+-------+------+--------+-------------+
|  0|      0|     0|       5|            5|
|  1|      0|     1|       3|            8|
|  2|      0|     2|       1|            9|
|  3|      1|     0|       2|           11|
|  4|      2|     0|       8|           19|
|  5|      2|     2|       8|           27|
+---+-------+------+--------+-------------+

val byOrderId = Window.partitionBy('orderID)
sales.withColumn("running_total_per_order", sum('orderQty).over(byOrderId)).show

+---+-------+------+--------+----------------------+
| id|orderID|prodID|orderQty|running_total_per_order|
+---+-------+------+--------+----------------------+
|  3|      1|     0|       2|                     2|
|  4|      2|     0|       8|                    16|
|  5|      2|     2|       8|                    16|
|  0|      0|     0|       5|                     9|
|  1|      0|     1|       3|                     9|
|  2|      0|     2|       1|                     9|
+---+-------+------+--------+----------------------+

byOrderId: org.apache.spark.sql.expressions.WindowSpec = org.apache.spark.sql.expressions.WindowSpec@647a9e69
```

row_number : Sequential number within window partition

```
empsalary.withColumn("row_number", row_number.over(byDepNameSalaryDesc)).show

+---------+-----+------+----------+
|  depName|empNo|salary|row_number|
+---------+-----+------+----------+
|  develop|    8|  6000|         1|
|  develop|   10|  5200|         2|
|  develop|   11|  5200|         3|
|  develop|    9|  4500|         4|
|  develop|    7|  4200|         5|
|    sales|    1|  5000|         1|
|    sales|   12|  5000|         2|
|    sales|    3|  4800|         3|
|    sales|    4|  4800|         4|
|    sales|    6|  4500|         5|
|personnel|    2|  3900|         1|
|personnel|    5|  3500|         2|
+---------+-----+------+----------+
```

Another example

```
val testDF = Seq(
(1525567980,28.9, "Ashland"),
(1525569900,28.4, "Bayfield"),
(1525571580,28.9, "Bayfield"),
(1525575120,28.9, "Ashland"),
(1525578780,28.0, "Ashland"),
(1525582320,28.0, "Bayfield"),
(1525585980,28.0, "Bayfield"),
(1525589580,28.0, "Ashland"),
(1525591440,28.4, "Bayfield"),
```

```
(1525593180,28.0, "Ashland"),
(1525596780,28.0, "Bayfield"),
(1525600380,28.9, "Bayfield")
).toDF("uxt", "Temp", "City")

testDF: org.apache.spark.sql.DataFrame = [uxt: int, Temp: double ... 1 more field]

testDF.show

+----------+----+--------+
|       uxt|Temp|    City|
+----------+----+--------+
|1525567980|28.9| Ashland|
|1525569900|28.4|Bayfield|
|1525571580|28.9|Bayfield|
|1525575120|28.9| Ashland|
|1525578780|28.0| Ashland|
|1525582320|28.0|Bayfield|
|1525585980|28.0|Bayfield|
|1525589580|28.0| Ashland|
|1525591440|28.4|Bayfield|
|1525593180|28.0| Ashland|
|1525596780|28.0|Bayfield|
|1525600380|28.9|Bayfield|
+----------+----+--------+

val testWithDates = testDF.withColumn("Date", to_timestamp(from_unixtime('uxt)))

testWithDates: org.apache.spark.sql.DataFrame = [uxt: int, Temp: double ... 2 more fields]

testWithDates.show

+----------+----+--------+-------------------+
|       uxt|Temp|    City|               Date|
+----------+----+--------+-------------------+
|1525567980|28.9| Ashland|2018-05-06 10:53:00|
|1525569900|28.4|Bayfield|2018-05-06 11:25:00|
|1525571580|28.9|Bayfield|2018-05-06 11:53:00|
|1525575120|28.9| Ashland|2018-05-06 12:52:00|
|1525578780|28.0| Ashland|2018-05-06 13:53:00|
|1525582320|28.0|Bayfield|2018-05-06 14:52:00|
|1525585980|28.0|Bayfield|2018-05-06 15:53:00|
|1525589580|28.0| Ashland|2018-05-06 16:53:00|
|1525591440|28.4|Bayfield|2018-05-06 17:24:00|
|1525593180|28.0| Ashland|2018-05-06 17:53:00|
|1525596780|28.0|Bayfield|2018-05-06 18:53:00|
|1525600380|28.9|Bayfield|2018-05-06 19:53:00|
+----------+----+--------+-------------------+

// Create a window or frame of 1 hr (3600 secs) prior to each row

val oneHrPriorFrame = Window.orderBy('uxt).rangeBetween(-3600, Window.currentRow)

oneHrPriorFrame: org.apache.spark.sql.expressions.WindowSpec = org.apache.spark.sql.expressions.WindowSpec@4da5a054

val byRange = testWithDates
    .withColumn("avgTemp", avg('Temp).over(oneHrPriorFrame))
    .withColumn("minTemp", min('Temp).over(oneHrPriorFrame))
    .withColumn("maxTemp", max('Temp).over(oneHrPriorFrame))
    .withColumn("rowCount", count('Temp).over(oneHrPriorFrame))
    .withColumn("frameStart", 'uxt - 3600)
    .withColumn("startDate", to_timestamp('frameStart))

byRange: org.apache.spark.sql.DataFrame = [uxt: int, Temp: double ... 8 more fields]

byRange.show

+----------+----+--------+-------------------+------------------+-------+-------+--------+----------+-------------------+
|       uxt|Temp|    City|               Date|           avgTemp|minTemp|maxTemp|rowCount|frameStart|          startDate|
+----------+----+--------+-------------------+------------------+-------+-------+--------+----------+-------------------+
|1525567980|28.9| Ashland|2018-05-06 10:53:00|              28.9|   28.9|   28.9|       1|1525564380|2018-05-06 09:53:00|
|1525569900|28.4|Bayfield|2018-05-06 11:25:00|             28.65|   28.4|   28.9|       2|1525566300|2018-05-06 10:25:00|
|1525571580|28.9|Bayfield|2018-05-06 11:53:00| 28.73333333333333|   28.4|   28.9|       3|1525567980|2018-05-06 10:53:00|
|1525575120|28.9| Ashland|2018-05-06 12:52:00|              28.9|   28.9|   28.9|       2|1525571520|2018-05-06 11:52:00|
|1525578780|28.0| Ashland|2018-05-06 13:53:00|              28.0|   28.0|   28.0|       1|1525575180|2018-05-06 12:53:00|
|1525582320|28.0|Bayfield|2018-05-06 14:52:00|              28.0|   28.0|   28.0|       2|1525578720|2018-05-06 13:52:00|
|1525585980|28.0|Bayfield|2018-05-06 15:53:00|              28.0|   28.0|   28.0|       1|1525582380|2018-05-06 14:53:00|
|1525589580|28.0| Ashland|2018-05-06 16:53:00|              28.0|   28.0|   28.0|       2|1525585980|2018-05-06 15:53:00|
|1525591440|28.4|Bayfield|2018-05-06 17:24:00|              28.2|   28.0|   28.4|       2|1525587840|2018-05-06 16:24:00|
|1525593180|28.0| Ashland|2018-05-06 17:53:00|28.133333333333336|   28.0|   28.4|       3|1525589580|2018-05-06 16:53:00|
|1525596780|28.0|Bayfield|2018-05-06 18:53:00|              28.0|   28.0|   28.0|       2|1525593180|2018-05-06 17:53:00|
|1525600380|28.9|Bayfield|2018-05-06 19:53:00|             28.45|   28.0|   28.9|       2|1525596780|2018-05-06 18:53:00|
+----------+----+--------+-------------------+------------------+-------+-------+--------+----------+-------------------+

// 2 hr window per city

val twoHrWindowByCity = Window.partitionBy('City).orderBy('uxt).rangeBetween(-7200, Window.currentRow)

twoHrWindowByCity: org.apache.spark.sql.expressions.WindowSpec = org.apache.spark.sql.expressions.WindowSpec@7a5d5e82

val byCity = testWithDates
    .withColumn("avgTemp", avg('Temp).over(twoHrWindowByCity))
    .withColumn("minTemp", min('Temp).over(twoHrWindowByCity))
    .withColumn("maxTemp", max('Temp).over(twoHrWindowByCity))
    .withColumn("rowCount", count('Temp).over(twoHrWindowByCity))
    .withColumn("frameStart", 'uxt - 3600)
    .withColumn("startDate", to_timestamp('frameStart))

byCity: org.apache.spark.sql.DataFrame = [uxt: int, Temp: double ... 8 more fields]

byCity.show
```

```
+----------+----+--------+-------------------+-------+-------+-------+--------+----------+-------------------+
|       uxt|Temp|    City|               Date|avgTemp|minTemp|maxTemp|rowCount|frameStart|          startDate|
+----------+----+--------+-------------------+-------+-------+-------+--------+----------+-------------------+
|1525567980|28.9| Ashland|2018-05-06 10:53:00|   28.9|   28.9|   28.9|       1|1525564380|2018-05-06 09:53:00|
|1525575120|28.9| Ashland|2018-05-06 12:52:00|   28.9|   28.9|   28.9|       2|1525571520|2018-05-06 11:52:00|
|1525578780|28.0| Ashland|2018-05-06 13:53:00|  28.45|   28.0|   28.9|       2|1525575180|2018-05-06 12:53:00|
|1525589580|28.0| Ashland|2018-05-06 16:53:00|   28.0|   28.0|   28.0|       1|1525585980|2018-05-06 15:53:00|
|1525593180|28.0| Ashland|2018-05-06 17:53:00|   28.0|   28.0|   28.0|       2|1525589580|2018-05-06 16:53:00|
|1525569900|28.4|Bayfield|2018-05-06 11:25:00|   28.4|   28.4|   28.4|       1|1525566300|2018-05-06 10:25:00|
|1525571580|28.9|Bayfield|2018-05-06 11:53:00|  28.65|   28.4|   28.9|       2|1525567980|2018-05-06 10:53:00|
|1525582320|28.0|Bayfield|2018-05-06 14:52:00|   28.0|   28.0|   28.0|       1|1525578720|2018-05-06 13:52:00|
|1525585980|28.0|Bayfield|2018-05-06 15:53:00|   28.0|   28.0|   28.0|       2|1525582380|2018-05-06 14:53:00|
|1525591440|28.4|Bayfield|2018-05-06 17:24:00|   28.2|   28.0|   28.4|       2|1525587840|2018-05-06 16:24:00|
|1525596780|28.0|Bayfield|2018-05-06 18:53:00|   28.2|   28.0|   28.4|       2|1525593180|2018-05-06 17:53:00|
|1525600380|28.9|Bayfield|2018-05-06 19:53:00|  28.45|   28.0|   28.9|       2|1525596780|2018-05-06 18:53:00|
+----------+----+--------+-------------------+-------+-------+-------+--------+----------+-------------------+
```