**Section 11: Apache Spark 2.x - Processing Data using Data Frames - Window Functions**

How to create window functions:

Windows have API’s such as Partition By and orderBy

For aggregations we can define the group by using partition by

For ranking or windowing we use partitionby and then order by

partitionBy is to group by data and orderBy is to sort the data to assign rank.

partitionBy or order by data returns WindowSpec object.

WindowSpec object needs to be passed to over with ranking and aggregate function.

employeePath = 'file:///F:/pySpark/data-master/data\_folder/hr\_db/employees/part-00000.csv'  
  
from pyspark.sql import SparkSession  
from pyspark import SQLContext  
# import sql functions  
from pyspark.sql.functions import \*  
  
# creating employees dataframe in spark  
  
spark =SparkSession.builder.master('local').appName('Windowing Function').getOrCreate()  
  
employees = spark.read.format('csv').option('sep', '\t'). \  
 schema('employee\_id INT, first\_name STRING, last\_name STRING, email STRING, phone\_number STRING, hire\_date STRING, job\_id STRING, '  
 'salary FLOAT, commission\_pct STRING, manager\_id STRING, department\_id STRING').load(employeePath)  
employees.select('employee\_id', 'department\_id', 'salary').show()

C:\Users\lenovo\AppData\Local\Programs\Python\Python37\python.exe C:/Users/lenovo/PycharmProjects/spark2demo/src/main/python/data\_frame\_windowing.py

20/09/17 10:04:06 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

+-------------+-----------+

|department\_id|sum(salary)|

+-------------+-----------+

| 30| 24900.0|

| 110| 20300.0|

| 100| 51600.0|

| 70| 10000.0|

| 90| 58000.0|

| 60| 28800.0|

| 40| 6500.0|

| 20| 19000.0|

| 10| 4400.0|

| 80| 304500.0|

| null| 7000.0|

| 50| 156400.0|

|  |  |
| --- | --- |
|  |  |

employees.select('employee\_id', 'department\_id', 'salary').groupBy('department\_id').sum('salary').show()

employees.select('employee\_id', 'department\_id', 'salary').groupBy('department\_id').agg(sum('salary').alias('salary\_expense')).show()

Main package is pyspark.sql.window

It has classes such as window and windowSpec.

Windows have API’s such as PartitionBy and OrderBy

These API’s return WindowSpec object.We can pass WindowSpec object over to functions such as rank(), dense\_rank(), sum()

Syntax: rank().over(spec) where spec = Window.paritionBy(‘ColumnName’)

Aggregation – sum, avg, min,max

Ranking – rank,dense\_rank, row\_number

Windowing –lead,lag

>>> from pyspark.sql import window

>>> help(window)

CLASSES

builtins.object

Window

WindowSpec

class Window(builtins.object)

| Utility functions for defining window in DataFrames.

|

| For example:

|

| >>> # ORDER BY date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

| >>> window = Window.orderBy("date").rowsBetween(Window.unboundedPreceding, Window.currentRow)

|

| >>> # PARTITION BY country ORDER BY date RANGE BETWEEN 3 PRECEDING AND 3 FOLLOWING

| >>> window = Window.orderBy("date").partitionBy("country").rangeBetween(-3, 3)

|

| .. note:: Experimental

|

| .. versionadded:: 1.4

|

| Static methods defined here:

|

| orderBy(\*cols)

| Creates a :class:`WindowSpec` with the ordering defined.

|

| .. versionadded:: 1.4

|

| orderBy(\*cols)

| Creates a :class:`WindowSpec` with the ordering defined.

|

| .. versionadded:: 1.4

|

| partitionBy(\*cols)

| Creates a :class:`WindowSpec` with the partitioning defined.

|

| .. versionadded:: 1.4

|

| rangeBetween(start, end)

| Creates a :class:`WindowSpec` with the frame boundaries defined,

| from `start` (inclusive) to `end` (inclusive).

|

| Both `start` and `end` are relative from the current row. For example,

| "0" means "current row", while "-1" means one off before the current row,

| and "5" means the five off after the current row.

|

| We recommend users use ``Window.unboundedPreceding``, ``Window.unboundedFollowing``,

| and ``Window.currentRow`` to specify special boundary values, rather than using integral

| values directly.

|

| :param start: boundary start, inclusive.

| The frame is unbounded if this is ``Window.unboundedPreceding``, or

| any value less than or equal to max(-sys.maxsize, -9223372036854775808).

| :param end: boundary end, inclusive.

| The frame is unbounded if this is ``Window.unboundedFollowing``, or

| any value greater than or equal to min(sys.maxsize, 9223372036854775807).

|

| .. versionadded:: 2.1

|>>> from pyspark.sql.window import \*

>>> orderItems = spark.read.json('file:///F:/pySpark/data-master/data\_folder/retail\_db\_json/order\_items')

>>> Window.partitionBy(orderItems.order\_item\_order\_id)

<pyspark.sql.window.WindowSpec object at 0x000001E5406970B8>

>>>

>>> spec = Window.partitionBy(orderItems.order\_item\_order\_id)

>>> from pyspark.sql import \*

>>> orderItems.withColumn('order\_revenue', sum('order\_item\_subtotal').over(spec)).show()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for +: 'int' and 'str'

>>> from pyspark.sql.functions import \*

>>> orderItems.withColumn('order\_revenue', sum('order\_item\_subtotal').over(spec)).show()

+-------------+-------------------+---------------------+------------------------+-------------------+-------------------+-----------------+

|order\_item\_id|order\_item\_order\_id|order\_item\_product\_id|order\_item\_product\_price|order\_item\_quantity|order\_item\_subtotal| order\_revenue|

+-------------+-------------------+---------------------+------------------------+-------------------+-------------------+-----------------+

| 83| 29| 1073| 199.99| 1| 199.99| 1109.85|

| 84| 29| 1014| 49.98| 5| 249.9| 1109.85|

| 85| 29| 403| 129.99| 1| 129.99| 1109.85|

| 86| 29| 403| 129.99| 1| 129.99| 1109.85|

| 87| 29| 1004| 399.98| 1| 399.98| 1109.85|

| 1161| 474| 1014| 49.98| 4| 199.92|774.8199999999999|

| 1162| 474| 502| 50.0| 3| 150.0|774.8199999999999|

| 1163| 474| 1073| 199.99| 1| 199.99|774.8199999999999|

| 1164| 474| 905| 24.99| 1| 24.99|774.8199999999999|

| 1165| 474| 1014| 49.98| 4| 199.92|774.8199999999999|

| 2395| 964| 627| 39.99| 4| 159.96|739.8800000000001|

| 2396| 964| 957| 299.98| 1| 299.98|739.8800000000001|

| 2397| 964| 977| 29.99| 5| 149.95|739.8800000000001|

| 2398| 964| 403| 129.99| 1| 129.99|739.8800000000001|

| 4170| 1677| 502| 50.0| 5| 250.0|649.9200000000001|

| 4171| 1677| 191| 99.99| 2| 199.98|649.9200000000001|

| 4172| 1677| 365| 59.99| 1| 59.99|649.9200000000001|

| 4173| 1677| 797| 17.99| 5| 89.95|649.9200000000001|

| 4174| 1677| 502| 50.0| 1| 50.0|649.9200000000001|

| 4514| 1806| 365| 59.99| 4| 239.96| 789.94|

+-------------+-------------------+---------------------+------------------------+-------------------+-------------------+-----------------+

only showing top 20 rows

>>> orderItems.withColumn('order\_revenue',round(sum('order\_item\_subtotal').over(spec), 2)).select('order\_item\_id', 'order\_item\_order\_id', 'order\_item\_subtotal', 'order\_revenue').show()

+-------------+-------------------+-------------------+-------------+

|order\_item\_id|order\_item\_order\_id|order\_item\_subtotal|order\_revenue|

+-------------+-------------------+-------------------+-------------+

| 83| 29| 199.99| 1109.85|

| 84| 29| 249.9| 1109.85|

| 85| 29| 129.99| 1109.85|

| 86| 29| 129.99| 1109.85|

| 87| 29| 399.98| 1109.85|

| 1161| 474| 199.92| 774.82|

| 1162| 474| 150.0| 774.82|

| 1163| 474| 199.99| 774.82|

| 1164| 474| 24.99| 774.82|

| 1165| 474| 199.92| 774.82|

| 2395| 964| 159.96| 739.88|

| 2396| 964| 299.98| 739.88|

| 2397| 964| 149.95| 739.88|

| 2398| 964| 129.99| 739.88|

| 4170| 1677| 250.0| 649.92|

| 4171| 1677| 199.98| 649.92|

| 4172| 1677| 59.99| 649.92|

| 4173| 1677| 89.95| 649.92|

| 4174| 1677| 50.0| 649.92|

| 4514| 1806| 239.96| 789.94|

+-------------+-------------------+-------------------+-------------+

only showing top 20 rows

==

Get top N products per day

Get daily product revenue

Use ranking function and rank the daily revenue

Once ranked filter in top n product revenue

Spark.conf.set(‘spark.sql.shuffle.partitions’, 2)

 Data Frame Operations - Creating Window  
Spec

Window have api’s like partitionBy and orderBy

For aggregation ,we can define the group by using partition by

For ranking or windowing ,we need to use partitionBy and orderBy.

partitionBy is used to group and orderBy is used to sort data, to assign rank.

partitionBy and orderBy returns windowSpec object

WindowSpec need to be passed pver ranking and aggregate functions

Let us see how to perform aggregations with in each group.

* We have functions such as sum, avg, min, max etc which can be used to aggregate the data.
* We need to create WindowSpec object using partitionBy to get aggregations with in each group.
* Some realistic use cases
  + Get average salary for each department and get all employee details who earn more than average salary
  + Get average revenue for each day and get all the orders who earn revenue more than average revenue
  + Get highest order revenue and get all the orders which have revenue more than 75% of the revenue

>>> employeePath = 'file:///F:/pySpark/data-master/data\_folder/hr\_db/employees/part-00000.csv'

>>> employees = spark.read.format('csv').option('sep', '\t'). \

... schema('employee\_id INT, first\_name STRING, last\_name STRING, email STRING, phone\_number STRING, hire\_date STRING, job\_id STRING, '

... 'salary FLOAT, commission\_pct STRING, manager\_id STRING, department\_id STRING').load(employeePath)

>>> employees.show()

+-----------+-----------+----------+--------+------------+----------+----------+-------+--------------+----------+-------------+

|employee\_id| first\_name| last\_name| email|phone\_number| hire\_date| job\_id| salary|commission\_pct|manager\_id|department\_id|

+-----------+-----------+----------+--------+------------+----------+----------+-------+--------------+----------+-------------+

| 100| Steven| King| SKING|515.123.4567|1987-06-17| AD\_PRES|24000.0| null| null| 90|

| 101| Neena| Kochhar|NKOCHHAR|515.123.4568|1989-09-21| AD\_VP|17000.0| null| 100| 90|

| 102| Lex| De Haan| LDEHAAN|515.123.4569|1993-01-13| AD\_VP|17000.0| null| 100| 90|

| 103| Alexander| Hunold| AHUNOLD|590.423.4567|1990-01-03| IT\_PROG| 9000.0| null| 102| 60|

| 104| Bruce| Ernst| BERNST|590.423.4568|1991-05-21| IT\_PROG| 6000.0| null| 103| 60|

| 105| David| Austin| DAUSTIN|590.423.4569|1997-06-25| IT\_PROG| 4800.0| null| 103| 60|

| 106| Valli| Pataballa|VPATABAL|590.423.4560|1998-02-05| IT\_PROG| 4800.0| null| 103| 60|

| 107| Diana| Lorentz|DLORENTZ|590.423.5567|1999-02-07| IT\_PROG| 4200.0| null| 103| 60|

| 108| Nancy| Greenberg|NGREENBE|515.124.4569|1994-08-17| FI\_MGR|12000.0| null| 101| 100|

| 109| Daniel| Faviet| DFAVIET|515.124.4169|1994-08-16|FI\_ACCOUNT| 9000.0| null| 108| 100|

| 110| John| Chen| JCHEN|515.124.4269|1997-09-28|FI\_ACCOUNT| 8200.0| null| 108| 100|

| 111| Ismael| Sciarra|ISCIARRA|515.124.4369|1997-09-30|FI\_ACCOUNT| 7700.0| null| 108| 100|

| 112|Jose Manuel| Urman| JMURMAN|515.124.4469|1998-03-07|FI\_ACCOUNT| 7800.0| null| 108| 100|

| 113| Luis| Popp| LPOPP|515.124.4567|1999-12-07|FI\_ACCOUNT| 6900.0| null| 108| 100|

| 114| Den| Raphaely|DRAPHEAL|515.127.4561|1994-12-07| PU\_MAN|11000.0| null| 100| 30|

| 115| Alexander| Khoo| AKHOO|515.127.4562|1995-05-18| PU\_CLERK| 3100.0| null| 114| 30|

| 116| Shelli| Baida| SBAIDA|515.127.4563|1997-12-24| PU\_CLERK| 2900.0| null| 114| 30|

| 117| Sigal| Tobias| STOBIAS|515.127.4564|1997-07-24| PU\_CLERK| 2800.0| null| 114| 30|

| 118| Guy| Himuro| GHIMURO|515.127.4565|1998-11-15| PU\_CLERK| 2600.0| null| 114| 30|

| 119| Karen|Colmenares|KCOLMENA|515.127.4566|1999-08-10| PU\_CLERK| 2500.0| null| 114| 30|

+-----------+-----------+----------+--------+------------+----------+----------+-------+--------------+----------+-------------+

only showing top 20 rows

>>> employees.select('employee\_id', 'department\_id', 'salary').show()

+-----------+-------------+-------+

|employee\_id|department\_id| salary|

+-----------+-------------+-------+

| 100| 90|24000.0|

| 101| 90|17000.0|

| 102| 90|17000.0|

| 103| 60| 9000.0|

| 104| 60| 6000.0|

| 105| 60| 4800.0|

| 106| 60| 4800.0|

| 107| 60| 4200.0|

| 108| 100|12000.0|

| 109| 100| 9000.0|

| 110| 100| 8200.0|

| 111| 100| 7700.0|

| 112| 100| 7800.0|

| 113| 100| 6900.0|

| 114| 30|11000.0|

| 115| 30| 3100.0|

| 116| 30| 2900.0|

| 117| 30| 2800.0|

| 118| 30| 2600.0|

| 119| 30| 2500.0|

|  |  |  |
| --- | --- | --- |
|  |  |  |

===

employeePath = 'file:///F:/pySpark/data-master/data\_folder/hr\_db/employees/part-00000.csv'  
  
from pyspark.sql import SparkSession  
from pyspark import SQLContext  
# import sql functions  
from pyspark.sql.functions import \*  
from pyspark.sql.window import \*  
from itertools import \*  
# creating employees dataframe in spark  
  
spark =SparkSession.builder.master('local').appName('Windowing Function').getOrCreate()  
  
employees = spark.read.format('csv').option('sep', '\t'). \  
 schema('employee\_id INT, first\_name STRING, last\_name STRING, email STRING, phone\_number STRING, hire\_date STRING, job\_id STRING, '  
 'salary FLOAT, commission\_pct STRING, manager\_id STRING, department\_id STRING').load(employeePath)  
  
spec =Window.partitionBy('department\_id')  
  
employees.select('employee\_id', 'department\_id', 'salary'). \  
 withColumn('salary\_expense', sum('salary').over(spec)).show()

spec =Window.partitionBy('department\_id')  
  
employees.select('employee\_id', 'department\_id', 'salary'). \  
 withColumn('salary\_expense', sum('salary').over(spec)).show()  
  
  
employees.select('employee\_id', 'department\_id', 'salary'). \  
 withColumn('salary\_expense', sum('salary').over(spec)).\  
 withColumn('max\_salary', max('salary').over(spec)). \  
 withColumn('min\_salary', min('salary').over(spec)). \  
 withColumn('avg\_salary', avg('salary').over(spec)). \  
 withColumn('salary\_pct',round((employees.salary/col('salary\_expense')) \* 100, 2)). \  
 sort('department\_id').show()

Data Frame Operations - Time Series  
Functions such as Lead, Lag et

We have functions like lead,lag first etc.

We need to create WindowSpec object using partitionBy and then orderBy for most of the windowing functions.

Lead and lag take column using which you want to get information based on partition and order columns,

Eg. The salary difference between current and next/previous employee with in each department.

Time series – revenue comparisons between two windows.

employeePath = 'file:///F:/pySpark/data-master/data\_folder/hr\_db/employees/part-00000.csv'  
  
from pyspark.sql import SparkSession  
from pyspark import SQLContext  
# import sql functions  
from pyspark.sql.functions import \*  
from pyspark.sql.window import \*  
from itertools import \*  
  
spark =SparkSession.builder.master('local').appName('Windowing Function').getOrCreate()  
  
employees = spark.read.format('csv').option('sep', '\t'). \  
 schema('employee\_id INT, first\_name STRING, last\_name STRING, email STRING, phone\_number STRING, hire\_date STRING, job\_id STRING, '  
 'salary FLOAT, commission\_pct STRING, manager\_id STRING, department\_id STRING').load(employeePath)  
  
spec = Window.partitionBy('department\_id').orderBy(employees.salary.desc())  
  
employees.select('employee\_id', 'department\_id', 'salary'). \  
 withColumn('next\_employee\_id', lead('employee\_id').over(spec)). \  
 sort('department\_id', employees.salary.desc()).show()

employees.select('employee\_id', 'department\_id', 'salary'). \  
 withColumn('previous\_employee\_id', lag('employee\_id').over(spec)). \  
 withColumn('previous salary', lag('salary').over(spec)). \  
 sort('department\_id', employees.salary.desc()).show()

:\Users\lenovo\AppData\Local\Programs\Python\Python37\python.exe C:/Users/lenovo/PycharmProjects/spark2demo/src/main/python/timeSeriesFunction.py

20/09/17 18:11:33 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

20/09/17 18:11:37 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.

+-----------+-------------+-------+----------------+-----------+

|employee\_id|department\_id| salary|next\_employee\_id|next salary|

+-----------+-------------+-------+----------------+-----------+

| 200| 10| 4400.0| null| null|

| 108| 100|12000.0| 109| 9000.0|

| 109| 100| 9000.0| 110| 8200.0|

| 110| 100| 8200.0| 112| 7800.0|

| 112| 100| 7800.0| 111| 7700.0|

| 111| 100| 7700.0| 113| 6900.0|

| 113| 100| 6900.0| null| null|

| 205| 110|12000.0| 206| 8300.0|

| 206| 110| 8300.0| null| null|

| 201| 20|13000.0| 202| 6000.0|

| 202| 20| 6000.0| null| null|

| 114| 30|11000.0| 115| 3100.0|

| 115| 30| 3100.0| 116| 2900.0|

| 116| 30| 2900.0| 117| 2800.0|

| 117| 30| 2800.0| 118| 2600.0|

| 118| 30| 2600.0| 119| 2500.0|

| 119| 30| 2500.0| null| null|

| 203| 40| 6500.0| null| null|

| 121| 50| 8200.0| 120| 8000.0|

| 120| 50| 8000.0| 122| 7900.0|

+-----------+-------------+-------+----------------+-----------+

only showing top 20 rows

+-----------+-------------+-------+--------------------+---------------+

|employee\_id|department\_id| salary|previous\_employee\_id|previous salary|

+-----------+-------------+-------+--------------------+---------------+

| 200| 10| 4400.0| null| null|

| 108| 100|12000.0| null| null|

| 109| 100| 9000.0| 108| 12000.0|

| 110| 100| 8200.0| 109| 9000.0|

| 112| 100| 7800.0| 110| 8200.0|

| 111| 100| 7700.0| 112| 7800.0|

| 113| 100| 6900.0| 111| 7700.0|

| 205| 110|12000.0| null| null|

| 206| 110| 8300.0| 205| 12000.0|

| 201| 20|13000.0| null| null|

| 202| 20| 6000.0| 201| 13000.0|

| 114| 30|11000.0| null| null|

| 115| 30| 3100.0| 114| 11000.0|

| 116| 30| 2900.0| 115| 3100.0|

| 117| 30| 2800.0| 116| 2900.0|

| 118| 30| 2600.0| 117| 2800.0|

| 119| 30| 2500.0| 118| 2600.0|

| 203| 40| 6500.0| null| null|

| 121| 50| 8200.0| null| null|

| 120| 50| 8000.0| 121| 8200.0|

+-----------+-------------+-------+--------------------+---------------+

only showing top 20 rows

Process finished with exit code 0

employees.select('employee\_id', 'department\_id', 'salary'). \  
 withColumn('first\_employee\_id', first('employee\_id').over(spec)). \  
 sort('department\_id', employees.salary.desc()).show()  
  
employees.select('employee\_id', 'department\_id', 'salary'). \  
 withColumn('first\_salary', first('salary').over(spec)). \  
 sort('department\_id', employees.salary.desc()).show()

spec = Window.partitionBy('department\_id').orderBy(employees.salary.desc()).rangeBetween(Window.unboundedPreceding, Window.unboundedFollowing )

Ranking function

Ranking function groupby data first and then sort the data for giving rank

Eg. Department partitionBy ,salary groupBy

Functions like rank,dense\_rank and row\_number

Assign rank to employees based on salary

Assign rank based on revenue each day or month

employeePath = 'file:///F:/pySpark/data-master/data\_folder/hr\_db/employees/part-00000.csv'  
  
from pyspark.sql import SparkSession  
from pyspark import SQLContext  
# import sql functions  
from pyspark.sql.functions import \*  
from pyspark.sql.window import \*  
from itertools import \*  
# creating employees dataframe in spark  
  
spark =SparkSession.builder.master('local').appName('Ranking').getOrCreate()  
  
employees = spark.read.format('csv').option('sep', '\t'). \  
 schema('employee\_id INT, first\_name STRING, last\_name STRING, email STRING, phone\_number STRING, hire\_date STRING, job\_id STRING, '  
 'salary FLOAT, commission\_pct STRING, manager\_id STRING, department\_id STRING').load(employeePath)  
  
spec = Window.partitionBy('department\_id'). \  
 orderBy(employees.salary.desc())  
  
  
employees.select('employee\_id','department\_id', 'salary'). \  
 withColumn('rank', rank().over(spec)). \  
 orderBy('department\_id',employees.salary.desc()).show()

employees.select('employee\_id','department\_id', 'salary'). \  
 withColumn('rank', rank().over(spec)). \  
 withColumn('\_dense\_rank', dense\_rank().over(spec)). \  
 withColumn('row\_number', row\_number().over(spec)). \  
 orderBy('department\_id',employees.salary.desc()).show(40)

C:\Users\lenovo\AppData\Local\Programs\Python\Python37\python.exe C:/Users/lenovo/PycharmProjects/spark2demo/src/main/python/ranking.py

20/09/17 20:01:55 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

20/09/17 20:01:58 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.

+-----------+-------------+-------+----+

|employee\_id|department\_id| salary|rank|

+-----------+-------------+-------+----+

| 200| 10| 4400.0| 1|

| 108| 100|12000.0| 1|

| 109| 100| 9000.0| 2|

| 110| 100| 8200.0| 3|

| 112| 100| 7800.0| 4|

| 111| 100| 7700.0| 5|

| 113| 100| 6900.0| 6|

| 205| 110|12000.0| 1|

| 206| 110| 8300.0| 2|

| 201| 20|13000.0| 1|

| 202| 20| 6000.0| 2|

| 114| 30|11000.0| 1|

| 115| 30| 3100.0| 2|

| 116| 30| 2900.0| 3|

| 117| 30| 2800.0| 4|

| 118| 30| 2600.0| 5|

| 119| 30| 2500.0| 6|

| 203| 40| 6500.0| 1|

| 121| 50| 8200.0| 1|

| 120| 50| 8000.0| 2|

| 122| 50| 7900.0| 3|

| 123| 50| 6500.0| 4|

| 124| 50| 5800.0| 5|

| 184| 50| 4200.0| 6|

| 185| 50| 4100.0| 7|

| 192| 50| 4000.0| 8|

| 193| 50| 3900.0| 9|

| 188| 50| 3800.0| 10|

| 137| 50| 3600.0| 11|

| 189| 50| 3600.0| 11|

| 141| 50| 3500.0| 13|

| 186| 50| 3400.0| 14|

| 133| 50| 3300.0| 15|

| 129| 50| 3300.0| 15|

| 194| 50| 3200.0| 17|

| 180| 50| 3200.0| 17|

| 125| 50| 3200.0| 17|

| 138| 50| 3200.0| 17|

| 181| 50| 3100.0| 21|

| 142| 50| 3100.0| 21|

+-----------+-------------+-------+----+

only showing top 40 rows

+-----------+-------------+-------+----+-----------+----------+

|employee\_id|department\_id| salary|rank|\_dense\_rank|row\_number|

+-----------+-------------+-------+----+-----------+----------+

| 200| 10| 4400.0| 1| 1| 1|

| 108| 100|12000.0| 1| 1| 1|

| 109| 100| 9000.0| 2| 2| 2|

| 110| 100| 8200.0| 3| 3| 3|

| 112| 100| 7800.0| 4| 4| 4|

| 111| 100| 7700.0| 5| 5| 5|

| 113| 100| 6900.0| 6| 6| 6|

| 205| 110|12000.0| 1| 1| 1|

| 206| 110| 8300.0| 2| 2| 2|

| 201| 20|13000.0| 1| 1| 1|

| 202| 20| 6000.0| 2| 2| 2|

| 114| 30|11000.0| 1| 1| 1|

| 115| 30| 3100.0| 2| 2| 2|

| 116| 30| 2900.0| 3| 3| 3|

| 117| 30| 2800.0| 4| 4| 4|

| 118| 30| 2600.0| 5| 5| 5|

| 119| 30| 2500.0| 6| 6| 6|

| 203| 40| 6500.0| 1| 1| 1|

| 121| 50| 8200.0| 1| 1| 1|

| 120| 50| 8000.0| 2| 2| 2|

| 122| 50| 7900.0| 3| 3| 3|

| 123| 50| 6500.0| 4| 4| 4|

| 124| 50| 5800.0| 5| 5| 5|

| 184| 50| 4200.0| 6| 6| 6|

| 185| 50| 4100.0| 7| 7| 7|

| 192| 50| 4000.0| 8| 8| 8|

| 193| 50| 3900.0| 9| 9| 9|

| 188| 50| 3800.0| 10| 10| 10|

| 137| 50| 3600.0| 11| 11| 11|

| 189| 50| 3600.0| 11| 11| 12|

| 141| 50| 3500.0| 13| 12| 13|

| 186| 50| 3400.0| 14| 13| 14|

| 133| 50| 3300.0| 15| 14| 16|

| 129| 50| 3300.0| 15| 14| 15|

| 194| 50| 3200.0| 17| 15| 20|

| 180| 50| 3200.0| 17| 15| 19|

| 125| 50| 3200.0| 17| 15| 17|

| 138| 50| 3200.0| 17| 15| 18|

| 181| 50| 3100.0| 21| 16| 22|

| 142| 50| 3100.0| 21| 16| 21|

+-----------+-------------+-------+----+-----------+----------+

only showing top 40 rows

Process finished with exit code 0