when / otherwise, alias, Asc / Desc, Cast, like, filter / where, Distinct / Drop Duplicate, orderby / sort union / unionAll, unionByName.

when() & otherwise()

- It is similar to SQL Case When, executes sequence of expressions until it matches the condition and returns a value when match.

```
from pyspark.sql.functions import when

df1 = df.select(df.id,df.name,when(condition=df.gender=='M',value='Male')\
          .when(condition=df.gender=='F',value='FeMale')\
          .otherwise('unknown').alias('gender'))

df1.show()
```

```
1  from pyspark.sql.functions import when
2
3
4  df1 = df.select(df.id,\
5                  df.name,\
6                  when(df.gender=='M','male')\
7                  .when(df.gender=='F','female')\
8                  .otherwise('unknown').alias('gender')\
9                 )
10 df1.show()
```

O/P

▸ (3) Spark Jobs

▸ df1: pyspark.sql.dataframe.DataFrame = [id: long, name: string ... 1 more

```
+---+------+-------+
| id|  name| gender|
+---+------+-------+
|  1|maheer|   male|
|  2|   Asi| female|
|  3|  abcd|unknown|
```

alias

- Provides alias to the column

```
data = [(1,'maheer','M',2000),(2,'wafa','M',4000),(3,'asi','F',3000)]

schema = ['id','name','gender','salary']

df = spark.createDataFrame(data,schema)

df.select(df.id.alias('emp_id'),df.name.alias('emp_name')).show()
```

O/P

```
+------+--------+----------+
|emp_id|emp_name|emp_salary|
+------+--------+----------+
|     1|  maheer|      2000|
|     2|    wafa|      4000|
|     3|     asi|      3000|
+------+--------+----------+
```

asc() & desc()

- Sort columns ascending or descending order.

```
data = [(1,'maheer','M',2000),(2,'wafa','M',4000),(3,'asi','F',3000)]

schema = ['id','name','gender','salary']

df = spark.createDataFrame(data,schema)

df.sort(df.name.desc()).show()
```

cast()

- convert the datatype

```
data = [(1,'maheer','M',2000),(2,'wafa','M',4000),(3,'asi','F',3000)]

schema = ['id','name','gender','salary']

df = spark.createDataFrame(data,schema)
df1 = df.select(df.salary.cast('int'))
df1.printSchema()
```

like()

- Similar to SQL LIKE expression

```
data = [(1,'maheer','M',2000),(2,'wafa','M',4000),(3,'asi','F',3000)]

schema = ['id','name','gender','salary']

df = spark.createDataFrame(data,schema)
df.filter(df.name.like('m%')).show()
```

filter() & where()

- PySpark filter() function is used to filter the rows from DataFrame based on the given condition or SQL expression.
- You can also use where() clause instead of the filter() if you are coming from an SQL background, both these functions operate exactly the same.

```
data = [(1,'maheer','M',2000),(2,'wafa','M',4000),(3,'asi','F',3000)]

schema = ['id','name','gender','salary']

df = spark.createDataFrame(data,schema)
df.where(df.gender == 'M').show()
df.filter(df.gender == 'M').show()
```

we can use any one where (or) filter

distinct() & dropDuplicates()

- PySpark distinct() function is used to remove the duplicate rows (all columns)
- dropDuplicates() is used to drop rows based on selected (one or multiple) columns.
- So basically, using these functions we can get distinct rows

```
data = [(1,'maheer','M',2000),(2,'wafa','M',4000),(2,'wafa','M',4000),(3,'asi','F',3000)]

schema = ['id','name','gender','salary']

df = spark.createDataFrame(data,schema)
df.distinct().show()
df.dropDuplicates().show()
df.dropDuplicates(['gender']).show()
```

orderBy() & sort()

- You can use either sort() or orderBy() function of PySpark DataFrame to sort DataFrame by ascending or descending order based on single or multiple columns.
- By default, sorting will happen in ascending order. We can explicitly mention ascending or descending using asc(), desc() functions.

```
data = [(1,'maheer','M',2000,'IT'),(2,'wafa','M',4000,'HR'),(3,'asi','F',3000,'Payroll'),(4,'sarfaraj','M',3000,'HR')
schema = ['id','name','gender','salary','dep']

df = spark.createDataFrame(data,schema)
df.sort('dep','salary').show()
df.sort(df.dep.desc(),df.salary.desc()).show()
df.orderBy(df.dep.desc(),df.salary.asc()).show()
df.orderBy('dep','salary').show()
```

order by and sort both same.

union() & unionAll()

- union() and unionAll() transformations are used to merge two or more DataFrame's of the same schema or structure.
- union() & unionAll() method merges two DataFrames and returns the new DataFrame with all rows from two Dataframes regardless of duplicate data.
- To remove duplicates use distinct() function

```
data = [(1,'maheer','M',2000,'IT'),(2,'wafa','M',4000,'HR'),(3,'asi','F',3000,'Payroll')]
schema = ['id','name','gender','salary','dep']

df1 = spark.createDataFrame(data,schema)
df2 = spark.createDataFrame(data,schema)
df1.show()
df2.show()
df1.unionAll(df2).show()
df1.union(df2).show()
```

* union will merge two table removing duplicate
* union all not remove duplicates while Merge table
* union and union all both are same No diff
* If we use df1.distinct it will remove dup

groupBy()

- Similar to SQL GROUP BY clause, PySpark groupBy() function is used to collect the identical data into groups on DataFrame and perform count, sum, avg, min, max functions on the grouped data

```
data = [(1,'maheer','M',5000,'IT'),\
        (2,'wafa','M',6000,'IT'),\
        (3,'asi','F',2500,'Payroll'),\
        (4,'sarfaraj','M',4000,'HR'),\
        (5,'pyarijaan','F',2000,'HR'),\
        (6,'Mahaboob','M',2000,'Payroll'),\
        (7,'ayesha','F',3000,'IT')]
schema = ['id','name','gender','salary','dep']
df = spark.createDataFrame(data,schema)
df.groupBy('dep').count().show()
df.groupBy('dep','gender').count().show()
df.groupBy('dep').min('salary').show()
df.groupBy('dep').max('salary').show()
df.groupBy('dep').avg('salary').show()
```

agg()

- PySpark Groupby agg() is used to calculate more than one aggregate (multiple aggregates) at a time on grouped DataFrame

```
data = [(1,'maheer','M',5000,'IT'),\
        (2,'wafa','M',6000,'IT'),\
        (3,'asi','F',2500,'Payroll'),\
        (4,'sarfaraj','M',4000,'HR'),\
        (5,'pyarijaan','F',2000,'HR'),\
        (6,'Mahaboob','M',2000,'Payroll'),\
        (7,'ayesha','F',3000,'IT')]
schema = ['id','name','gender','salary','dep']
df = spark.createDataFrame(data,schema)
df.groupBy('dep').count().show()
from pyspark.sql.functions import count,min,max
df.groupBy('dep').agg(count('*').alias('countOfEmps'),\
                      min('salary').alias('minSal'),\
                      max('salary').alias('maxSal')).show()
```

```
+---+---------+------+------+-------+
| id|     name|gender|salary|    dep|
+---+---------+------+------+-------+
|  1|   maheer|     M|  5000|     IT|
|  2|     wafa|     M|  6000|     IT|
|  3|      asi|     F|  2500|Payroll|
|  4| sarfaraj|     M|  4000|     HR|
|  5|pyarijaan|     F|  2000|     HR|
|  6| Mahaboob|     M|  2000|Payroll|
|  7|   ayesha|     F|  3000|     IT|
+---+---------+------+------+-------+
```

o/p

```
+-------+----------+------+------+
|    dep|countOfEmps|minSal|maxSal|
+-------+----------+------+------+
|     IT|         3|  3000|  6000|
|     HR|         2|  2000|  4000|
|Payroll|         2|  2000|  2500|
+-------+----------+------+------+
```

Comparison table

Method	Matching Basis	Missing Columns Allowed	Duplicates
`union`	Schema (exact match)	No	Retained
`unionAll`	Same as `union` (deprecated)	No	Retained
`unionByName`	Column names	Yes (with parameter)	Retained

unionByName()

- unionByName() lets you to merge/union two DataFrames with a different number of columns (different schema) by passing allowMissingColumns with the value true

```
data1 = [(1,'maheer','male')]
schema1 = ['id','name','gender']

data2 = [(1,'maheer',2000)]
schema2 = ['id','name','salary']

df1 = spark.createDataFrame(data1,schema1)
df2 = spark.createDataFrame(data2,schema2)

df1.union(df2).show()

df1.unionByName(allowMissingColumns=True,other=df2).show()
```

Sabry ?

```
+---+------+---+
| id|  name|age|
+---+------+---+
|  1|maheer| 31|
+---+------+---+
```

o/p

```
+---+------+------+
| id|  name|salary|
+---+------+------+
|  1|maheer|  2000|
+---+------+------+
```

```
+---+------+----+------+
| id|  name| age|salary|
+---+------+----+------+
|  1|maheer|  31|  null|
|  1|maheer|null|  2000|
+---+------+----+------+
```