==========

To Select 2 columns from table “df”

==========

spark.sql("select id, tdate from df").show()

```
+---+----------+
| id|     tdate|
+---+----------+
|  0|06-26-2011|
|  1|05-26-2011|
|  2|06-01-2011|
|  3|06-05-2011|
|  4|12-17-2011|
|  5|02-14-2011|
|  6|06-05-2011|
|  7|12-17-2011|
|  8|02-14-2011|
+---+----------+
```

==========

To select columns where category is “Exercise”

==========

spark.sql("select * from df where category = 'Exercise'").show()

```
+---+----------+------+--------+-------------+-------+
| id|     tdate|amount|category|      product|spendby|
+---+----------+------+--------+-------------+-------+
|  0|06-26-2011| 300.4|Exercise| GymnasticsPro|   cash|
|  2|06-01-2011| 300.4|Exercise|Gymnastics Pro|   cash|
|  6|06-05-2011| 100.0|Exercise|        Rings| credit|
+---+----------+------+--------+-------------+-------+
```

==========

To select columns where category is “Exercise” and spendby is cash and select only 4 columns (id, tdate, category, and spendby)

==========

spark.sql("select id, tdate, category, spendby from df where category = 'Exercise' and spendby='cash'").show()

```
+---+----------+--------+-------+
| id|     tdate|category|spendby|
+---+----------+--------+-------+
|  0|06-26-2011|Exercise|   cash|
|  2|06-01-2011|Exercise|   cash|
+---+----------+--------+-------+
```

==========

To select columns where category is “Exercise” and “Gymnastics”

[Here IN operator will come in use]

==========

spark.sql("select * from df where category in ('Exercise' , 'Gymnastics')").show()

```
+---+----------+------+----------+-------------+-------+
| id|     tdate|amount|  category|      product|spendby|
+---+----------+------+----------+-------------+-------+
|  0|06-26-2011| 300.4|  Exercise| GymnasticsPro|   cash|
|  2|06-01-2011| 300.4|  Exercise|Gymnastics Pro|   cash|
|  3|06-05-2011| 100.0|Gymnastics|         Rings| credit|
|  5|02-14-2011| 200.0|Gymnastics|          NULL|   cash|
|  6|06-05-2011| 100.0|  Exercise|         Rings| credit|
|  8|02-14-2011| 200.0|Gymnastics|          NULL|   cash|
+---+----------+------+----------+-------------+-------+
```

==========

To filter columns in such a way that it shows product as “Gymnastics”

==========

spark.sql("select * from df where product like '%Gymnastics%'").show()

```
+---+----------+------+--------+-------------+-------+
| id|     tdate|amount|category|      product|spendby|
+---+----------+------+--------+-------------+-------+
|  0|06-26-2011| 300.4|Exercise| GymnasticsPro|   cash|
|  2|06-01-2011| 300.4|Exercise|Gymnastics Pro|   cash|
+---+----------+------+--------+-------------+-------+
```

==========

To filter columns in such a way that it shows category not equal to “Excercise”

==========

spark.sql("select * from df where category != 'Exercise'").show()

```
+---+----------+------+-------------+-------------+-------+
| id|     tdate|amount|     category|      product|spendby|
+---+----------+------+-------------+-------------+-------+
|  1|05-26-2011| 200.0|Exercise Band|Weightlifting| credit|
|  3|06-05-2011| 100.0|   Gymnastics|        Rings| credit|
|  4|12-17-2011| 300.0|  Team Sports|        Field|   cash|
|  5|02-14-2011| 200.0|   Gymnastics|         NULL|   cash|
|  7|12-17-2011| 300.0|  Team Sports|        Field|   cash|
|  8|02-14-2011| 200.0|   Gymnastics|         NULL|   cash|
+---+----------+------+-------------+-------------+-------+
```

==========

To filter columns in such a way that it shows category not equal to “Exercise and Gymnastics”

[Multi value]

==========

spark.sql("select * from df where category not in ('Exercise', 'Gymnastics')").show()

```
+---+----------+------+-------------+-------------+-------+
| id|     tdate|amount|     category|      product|spendby|
+---+----------+------+-------------+-------------+-------+
|  1|05-26-2011| 200.0|Exercise Band|Weightlifting| credit|
|  4|12-17-2011| 300.0|  Team Sports|        Field|   cash|
|  7|12-17-2011| 300.0|  Team Sports|        Field|   cash|
+---+----------+------+-------------+-------------+-------+
```

==========

To select product column where “Null” value

==========

spark.sql("select * from df where product is null").show()

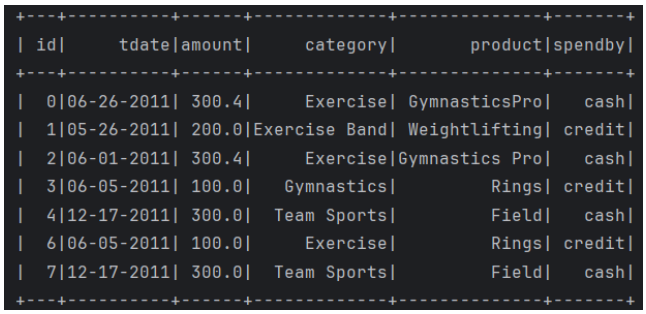
```
+---+----------+------+----------+-------+-------+
| id|     tdate|amount|  category|product|spendby|
+---+----------+------+----------+-------+-------+
|  5|02-14-2011| 200.0|Gymnastics|   NULL|   cash|
|  8|02-14-2011| 200.0|Gymnastics|   NULL|   cash|
+---+----------+------+----------+-------+-------+
```

==========

To select product column where value is not “Null”

==========

spark.sql("select * from df where product is not null").show()



id	tdate	amount	category	product	spendby
0	06-26-2011	300.4	Exercise	GymnasticsPro	cash
1	05-26-2011	200.0	Exercise Band	Weightlifting	credit
2	06-01-2011	300.4	Exercise	Gymnastics Pro	cash
3	06-05-2011	100.0	Gymnastics	Rings	credit
4	12-17-2011	300.0	Team Sports	Field	cash
6	06-05-2011	100.0	Exercise	Rings	credit
7	12-17-2011	300.0	Team Sports	Field	cash

==========

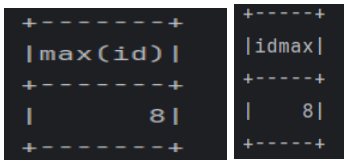
To select “Max of Id”

==========

spark.sql("select max(id) from df").show()

or we can rename it as well

spark.sql("select max(id) as idmax from df").show()



max(id)
8

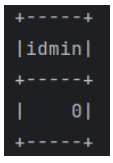
idmax
8

==========

To select “Min of Id”

==========

spark.sql("select min(id) as idmin from df").show()



idmin
0

==========

To get count of rows

==========

spark.sql("select count(1) from df").show()

we can also use

spark.sql("select count(*) from df").show()

```
+--------+
|count(1)|
+--------+
|       9|
+--------+
```

==========

To get extra row on right side of table as “Status” like (for cash as 1 and for credit as 0) depending on the “Spendby” column:

Conditional Statement

==========

spark.sql("select *, case when spendby='cash' then 1 else 0 end as status from df").show()

```
+---+----------+------+-------------+--------------+-------+------+
| id|     tdate|amount|     category|       product|spendby|status|
+---+----------+------+-------------+--------------+-------+------+
|  0|06-26-2011| 300.4|     Exercise| GymnasticsPro|   cash|     1|
|  1|05-26-2011| 200.0|Exercise Band| Weightlifting| credit|     0|
|  2|06-01-2011| 300.4|     Exercise|Gymnastics Pro|   cash|     1|
|  3|06-05-2011| 100.0|   Gymnastics|         Rings| credit|     0|
|  4|12-17-2011| 300.0|  Team Sports|         Field|   cash|     1|
|  5|02-14-2011| 200.0|   Gymnastics|          NULL|   cash|     1|
|  6|06-05-2011| 100.0|     Exercise|         Rings| credit|     0|
|  7|12-17-2011| 300.0|  Team Sports|         Field|   cash|     1|
|  8|02-14-2011| 200.0|   Gymnastics|          NULL|   cash|     1|
+---+----------+------+-------------+--------------+-------+------+
```

==========

Concat Two Columns

==========

To get Extra Columns as “Condata” by concatenating two columns with “-“

spark.sql("select id, category, concat(id, '-', category) as condata from df").show()

id	category	condata
0	Exercise	0-Exercise
1	Exercise Band	1-Exercise Band
2	Exercise	2-Exercise
3	Gymnastics	3-Gymnastics
4	Team Sports	4-Team Sports
5	Gymnastics	5-Gymnastics
6	Exercise	6-Exercise
7	Team Sports	7-Team Sports
8	Gymnastics	8-Gymnastics

==========

To concat 3 columns

==========

spark.sql("select id, category, product, concat(id, '-', category, '-', product) as condata from df").show()

id	category	product	condata
0	Exercise	GymnasticsPro	0-Exercise-Gymnas...
1	Exercise Band	Weightlifting	1-Exercise Band-W...
2	Exercise	Gymnastics Pro	2-Exercise-Gymnas...
3	Gymnastics	Rings	3-Gymnastics-Rings
4	Team Sports	Field	4-Team Sports-Field
5	Gymnastics	NULL	NULL
6	Exercise	Rings	6-Exercise-Rings
7	Team Sports	Field	7-Team Sports-Field
8	Gymnastics	NULL	NULL

==========

To concat multiple columns

==========

spark.sql("select id, category, product, concat_ws('-', id, category, product) as condata from df").show()

```
+---+------------+-------------+-------------------+
| id|    category|      product|            condata|
+---+------------+-------------+-------------------+
|  0|    Exercise|GymnasticsPro|0-Exercise-Gymnas...|
|  1|Exercise Band|Weightlifting|1-Exercise Band-W...|
|  2|    Exercise|Gymnastics Pro|2-Exercise-Gymnas...|
|  3|  Gymnastics|        Rings|  3-Gymnastics-Rings|
|  4| Team Sports|        Field| 4-Team Sports-Field|
|  5|  Gymnastics|         NULL|        5-Gymnastics|
|  6|    Exercise|        Rings|    6-Exercise-Rings|
|  7| Team Sports|        Field| 7-Team Sports-Field|
|  8|  Gymnastics|         NULL|        8-Gymnastics|
+---+------------+-------------+-------------------+
```

==========

To get Category column in Lower case

==========

spark.sql("select category, lower(category) as lower from df").show()

```
+------------+------------+
|    category|       lower|
+------------+------------+
|    Exercise|    exercise|
|Exercise Band|exercise band|
|    Exercise|    exercise|
|  Gymnastics|  gymnastics|
| Team Sports| team sports|
|  Gymnastics|  gymnastics|
|    Exercise|    exercise|
| Team Sports| team sports|
|  Gymnastics|  gymnastics|
+------------+------------+
```

==========

To get Category column in Upper case

==========

spark.sql("select category, upper(category) as upper from df").show()

```
+-------------+-------------+
|     category|        upper|
+-------------+-------------+
|     Exercise|     EXERCISE|
|Exercise Band|EXERCISE BAND|
|     Exercise|     EXERCISE|
|   Gymnastics|   GYMNASTICS|
|  Team Sports|  TEAM SPORTS|
|   Gymnastics|   GYMNASTICS|
|     Exercise|     EXERCISE|
|  Team Sports|  TEAM SPORTS|
|   Gymnastics|   GYMNASTICS|
+-------------+-------------+
```

==========

CEIL Function: It represents rounding of value with upper value

==========

spark.sql("select amount, ceil(amount) as ceil from df").show()

```
+------+----+
|amount|ceil|
+------+----+
| 300.4| 301|
| 200.0| 200|
| 300.4| 301|
| 100.0| 100|
| 300.0| 300|
| 200.0| 200|
| 100.0| 100|
| 300.0| 300|
| 200.0| 200|
+------+----+
```

==========

Round value Function: It represents rounding of value with nearest integer value

==========

spark.sql("select amount, round(amount) as round from df").show()

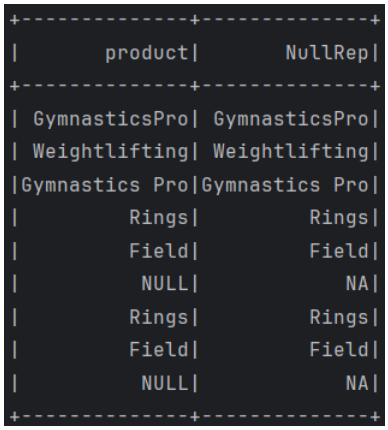
```
+------+-----+
|amount|round|
+------+-----+
| 300.4|300.0|
| 200.0|200.0|
| 300.4|300.0|
| 100.0|100.0|
| 300.0|300.0|
| 200.0|200.0|
| 100.0|100.0|
| 300.0|300.0|
| 200.0|200.0|
+------+-----+
```

==========

To Replace Nulls in table with “NA”

==========

spark.sql("select product, coalesce(product, 'NA') as NullRep from df").show()



```
+--------------+--------------+
|       product|       NullRep|
+--------------+--------------+
| GymnasticsPro| GymnasticsPro|
| Weightlifting| Weightlifting|
|Gymnastics Pro|Gymnastics Pro|
|         Rings|         Rings|
|         Field|         Field|
|          NULL|            NA|
|         Rings|         Rings|
|         Field|         Field|
|          NULL|            NA|
+--------------+--------------+
```

==========

To Trim the space “ ___Gymnastics___” . Only Front and Back Spaces will be removed

==========

spark.sql("select trim(product) from df").show()

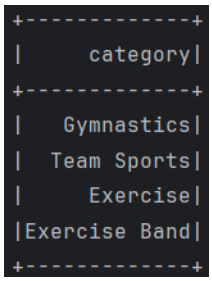
“No Example in this table”

==========

Distinct Function---Taking the unique values

==========

spark.sql("select distinct category from df").show()



```
+--------------+
|      category|
+--------------+
|    Gymnastics|
|   Team Sports|
|      Exercise|
|Exercise Band|
+--------------+
```

We can perform Distinct on multiple columns

spark.sql("select distinct category, spendby from df").show()

```
+------------+-------+
|    category|spendby|
+------------+-------+
|  Gymnastics|   cash|
|    Exercise|   cash|
|    Exercise| credit|
| Team Sports|   cash|
|Exercise Band| credit|
|  Gymnastics| credit|
+------------+-------+
```

==========

SUBSTRING –Suppose we have to perform substring on product Columns and we need first 10 characters of particular word

==========

spark.sql("select substring(product,1, 10) as SUBST from df").show()

```
+----------+
|     SUBST|
+----------+
|Gymnastics|
|Weightlift|
|Gymnastics|
|     Rings|
|     Field|
|      NULL|
|     Rings|
|     Field|
|      NULL|
+----------+
```

We can also put as per requirement. Lets say we have to choose from 3rd character onwards

Eg-- spark.sql("select substring(product,3, 10) as SUBST from df").show()

==========

SPLIT OF Function:

==========

spark.sql("select product, split(product, ' ')[0] as split from df").show()

```
+--------------+-------------+
|       product|        split|
+--------------+-------------+
| GymnasticsPro|GymnasticsPro|
| Weightlifting|Weightlifting|
|Gymnastics Pro|   Gymnastics|
|         Rings|        Rings|
|         Field|        Field|
|          NULL|         NULL|
|         Rings|        Rings|
|         Field|        Field|
|          NULL|         NULL|
+--------------+-------------+
```

==========

UNION ALL of Two DataFrames—Combining of two data frames vertically and it will not remove duplicites

==========

spark.sql("select * from df union all select * from df1").show()

```
+---+----------+------+-------------+--------------+-------+
| id|     tdate|amount|     category|       product|spendby|
+---+----------+------+-------------+--------------+-------+
|  0|06-26-2011| 300.4|     Exercise| GymnasticsPro|   cash|
|  1|05-26-2011| 200.0|Exercise Band| Weightlifting| credit|
|  2|06-01-2011| 300.4|     Exercise|Gymnastics Pro|   cash|
|  3|06-05-2011| 100.0|   Gymnastics|         Rings| credit|
|  4|12-17-2011| 300.0|  Team Sports|         Field|   cash|
|  5|02-14-2011| 200.0|   Gymnastics|          NULL|   cash|
|  6|06-05-2011| 100.0|     Exercise|         Rings| credit|
|  7|12-17-2011| 300.0|  Team Sports|         Field|   cash|
|  8|02-14-2011| 200.0|   Gymnastics|          NULL|   cash|
|  4|12-17-2011| 300.0|  Team Sports|         Field|   cash|
|  5|02-14-2011| 200.0|   Gymnastics|          NULL|   cash|
|  6|02-14-2011| 200.0|       Winter|          NULL|   cash|
|  7|02-14-2011| 200.0|       Winter|          NULL|   cash|
+---+----------+------+-------------+--------------+-------+
```

==========

UNION of Two DataFrames—Combining of two data frames vertically and it will remove duplicities

==========

spark.sql("select * from df union select * from df1").show()

```
+---+----------+------+------------+-------------+-------+
| id|     tdate|amount|     category|      product|spendby|
+---+----------+------+------------+-------------+-------+
|  7|12-17-2011| 300.0|  Team Sports|        Field|   cash|
|  0|06-26-2011| 300.4|     Exercise|GymnasticsPro|   cash|
|  6|06-05-2011| 100.0|     Exercise|        Rings| credit|
|  3|06-05-2011| 100.0|   Gymnastics|        Rings| credit|
|  2|06-01-2011| 300.4|     Exercise|Gymnastics Pro|   cash|
|  4|12-17-2011| 300.0|  Team Sports|        Field|   cash|
|  1|05-26-2011| 200.0|Exercise Band|Weightlifting| credit|
|  8|02-14-2011| 200.0|   Gymnastics|         NULL|   cash|
|  5|02-14-2011| 200.0|   Gymnastics|         NULL|   cash|
|  6|02-14-2011| 200.0|       Winter|         NULL|   cash|
|  7|02-14-2011| 200.0|       Winter|         NULL|   cash|
+---+----------+------+------------+-------------+-------+
```

==========

To get “Total amount collected for each Category”

==========

spark.sql("select category, sum(amount) as sum from df group by category ").show()

```
+-------------+-----------+
|     category|sum(amount)|
+-------------+-----------+
|   Gymnastics|      500.0|
|  Team Sports|      600.0|
|     Exercise|      700.8|
|Exercise Band|      200.0|
+-------------+-----------+
```

==========

Grouping Two Columns—To find total amount spent on category and spendby

==========

spark.sql("select category,spendby, sum(amount) as sum from df group by category,spendby ").show()

```
+-------------+-------+-----+
|     category|spendby|  sum|
+-------------+-------+-----+
|   Gymnastics|   cash|400.0|
|     Exercise|   cash|600.8|
|     Exercise| credit|100.0|
|  Team Sports|   cash|600.0|
|Exercise Band| credit|200.0|
|   Gymnastics| credit|100.0|
+-------------+-------+-----+
```

==========

To count the number of rows counted in above scenario

==========

spark.sql("select category,spendby, sum(amount) as sum, count(amount) as CNT from df group by category,spendby ").show()

```
+-------------+-------+-----+---+
|     category|spendby|  sum|CNT|
+-------------+-------+-----+---+
|   Gymnastics|   cash|400.0|  2|
|     Exercise|   cash|600.8|  2|
|     Exercise| credit|100.0|  1|
|  Team Sports|   cash|600.0|  2|
|Exercise Band| credit|200.0|  1|
|   Gymnastics| credit|100.0|  1|
+-------------+-------+-----+---+
```

==========

To get MAX amount of every category column

==========

spark.sql("select category, max(amount) as max from df group by category ").show()

```
+-------------+-----+
|     category|  max|
+-------------+-----+
|   Gymnastics|200.0|
|  Team Sports|300.0|
|     Exercise|300.4|
|Exercise Band|200.0|
+-------------+-----+
```

==========

To get MIN amount of every category column

==========

spark.sql("select category, min(amount) as min from df group by category ").show()

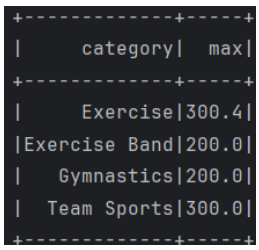
```
+-------------+-----+
|     category|  min|
+-------------+-----+
|   Gymnastics|100.0|
|  Team Sports|300.0|
|     Exercise|100.0|
|Exercise Band|200.0|
+-------------+-----+
```

==========

To get the category in proper ORDER –we can perform order by

==========

spark.sql("select category, max(amount) as max from df group by category order by category ").show()



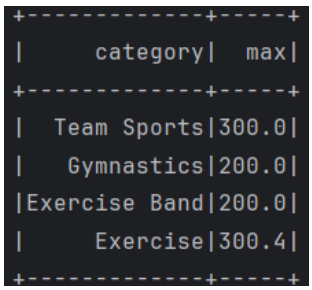
```
+-------------+-----+
|     category|  max|
+-------------+-----+
|     Exercise|300.4|
|Exercise Band|200.0|
|   Gymnastics|200.0|
|  Team Sports|300.0|
+-------------+-----+
```

==========

To make it Descending

==========

spark.sql("select category, max(amount) as max from df group by category order by category desc ").show()



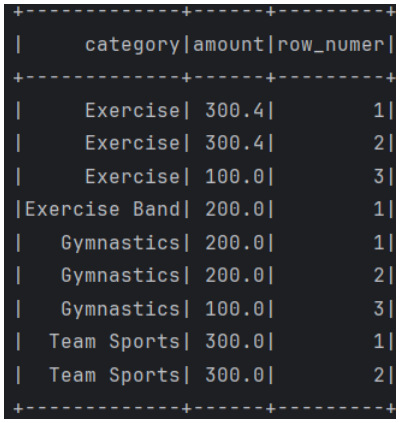
```
+-------------+-----+
|     category|  max|
+-------------+-----+
|  Team Sports|300.0|
|   Gymnastics|200.0|
|Exercise Band|200.0|
|     Exercise|300.4|
+-------------+-----+
```

==========

WINDOWS ROW OPERATIONS—means sections of Data

==========

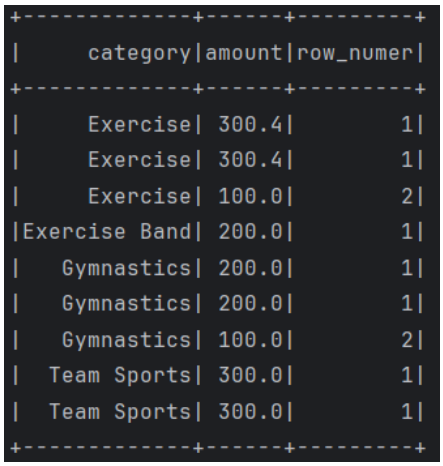
spark.sql("select category, amount, row_number() OVER (partition by category order by amount desc) AS row_numer from df ").show()



category	amount	row_numer
Exercise	300.4	1
Exercise	300.4	2
Exercise	100.0	3
Exercise Band	200.0	1
Gymnastics	200.0	1
Gymnastics	200.0	2
Gymnastics	100.0	3
Team Sports	300.0	1
Team Sports	300.0	2

Similarly we can use dense rank() in place of row_number----It will rank the same values

spark.sql("select category, amount, dense_rank() OVER (partition by category order by amount desc) AS row_numer from df ").show()



category	amount	row_numer
Exercise	300.4	1
Exercise	300.4	1
Exercise	100.0	2
Exercise Band	200.0	1
Gymnastics	200.0	1
Gymnastics	200.0	1
Gymnastics	100.0	2
Team Sports	300.0	1
Team Sports	300.0	1

==========

Only RANK case

==========

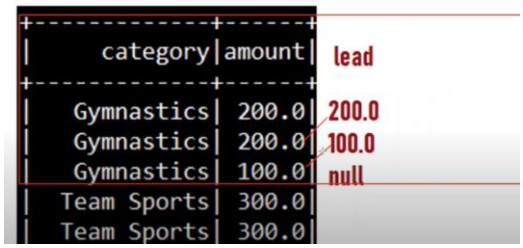
spark.sql("select category, amount, rank() OVER (partition by category order by amount desc) AS row_numer from df ").show()

category	amount	row_numer
Exercise	300.4	1
Exercise	300.4	1
Exercise	100.0	3
Exercise Band	200.0	1
Gymnastics	200.0	1
Gymnastics	200.0	1
Gymnastics	100.0	3
Team Sports	300.0	1
Team Sports	300.0	1

==========

Window LEAD Functions (Ex given below)—First value will come up and 2nd or last value becomes null

==========



spark.sql("select category, amount, lead (amount) OVER (partition by category order by amount desc) as lead from df ").show()

category	amount	lead
Exercise	300.4	300.4
Exercise	300.4	100.0
Exercise	100.0	NULL
Exercise Band	200.0	NULL
Gymnastics	200.0	200.0
Gymnastics	200.0	100.0
Gymnastics	100.0	NULL
Team Sports	300.0	300.0
Team Sports	300.0	NULL

==========

WINDOWs Lag Function—Means 1 step down –Data will come down

==========

spark.sql("select category, amount, lag (amount) OVER (partition by category order by amount desc) as lag from df ").show()

```
+-------------+------+-----+
|     category|amount|  lag|
+-------------+------+-----+
|     Exercise| 300.4| NULL|
|     Exercise| 300.4|300.4|
|     Exercise| 100.0|300.4|
|Exercise Band| 200.0| NULL|
|   Gymnastics| 200.0| NULL|
|   Gymnastics| 200.0|200.0|
|   Gymnastics| 100.0|200.0|
|  Team Sports| 300.0| NULL|
|  Team Sports| 300.0|300.0|
+-------------+------+-----+
```

==========

Having Functions: will tell you how many times rows gets duplicated'

==========

spark.sql("select category, count(category) as CNT from df group by category having count(category)>1 ").show()

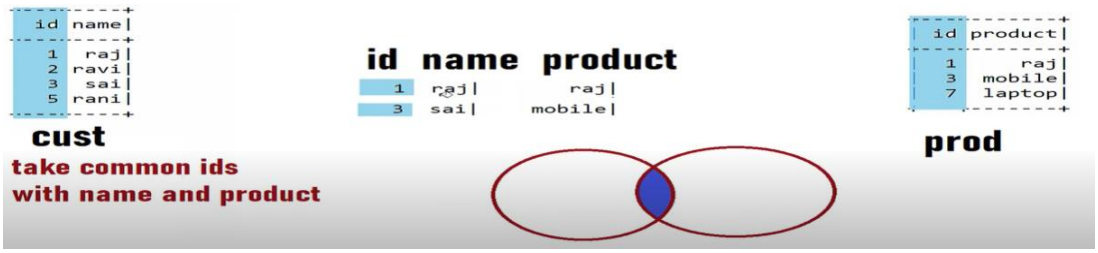
```
+-----------+---+
|   category|CNT|
+-----------+---+
| Gymnastics|  3|
|Team Sports|  2|
|   Exercise|  3|
+-----------+---+
```

Joins

==========

Inner Join

==========



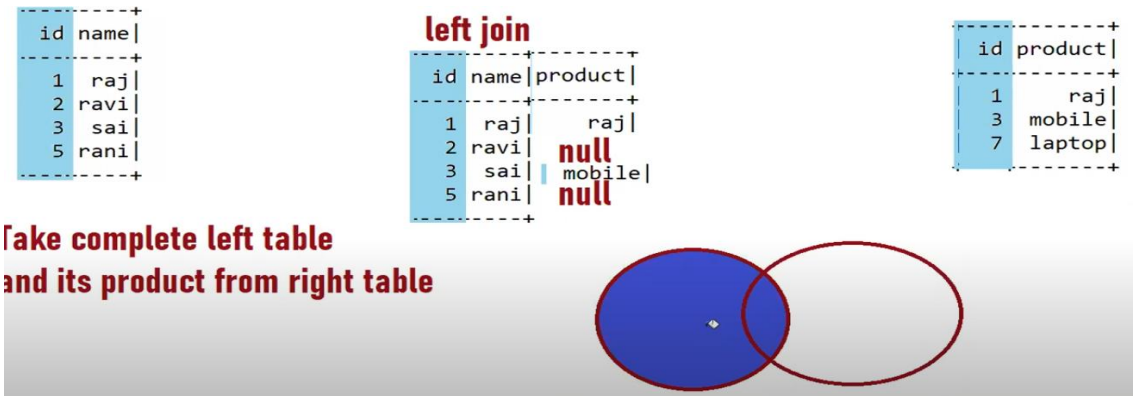
spark.sql("select a.*, b.product from cust a join prod b on a.id=b.id ").show()

```
+---+----+-------+
| id|name|product|
+---+----+-------+
|  1| raj|  mouse|
|  3| sai| mobile|
+---+----+-------+
```

==========

Left Join

==========



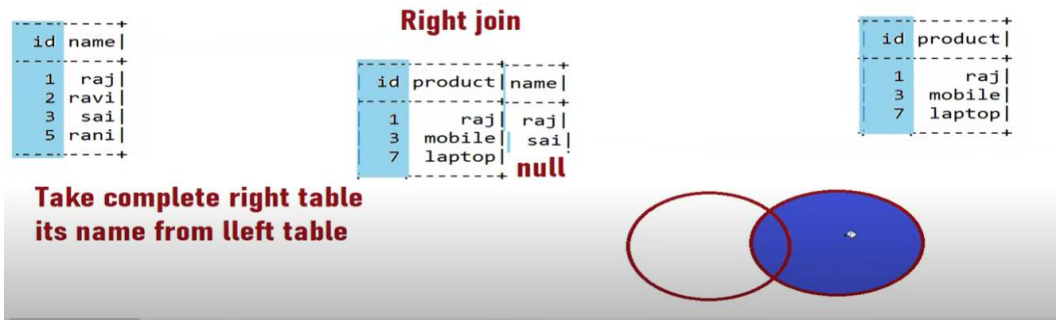
spark.sql("select a.*, b.product from cust a left join prod b on a.id=b.id ").show()

```
+---+----+-------+
| id|name|product|
+---+----+-------+
|  5|rani|   NULL|
|  1| raj|  mouse|
|  3| sai| mobile|
|  2|ravi|   NULL|
+---+----+-------+
```

==========

Right Join

==========



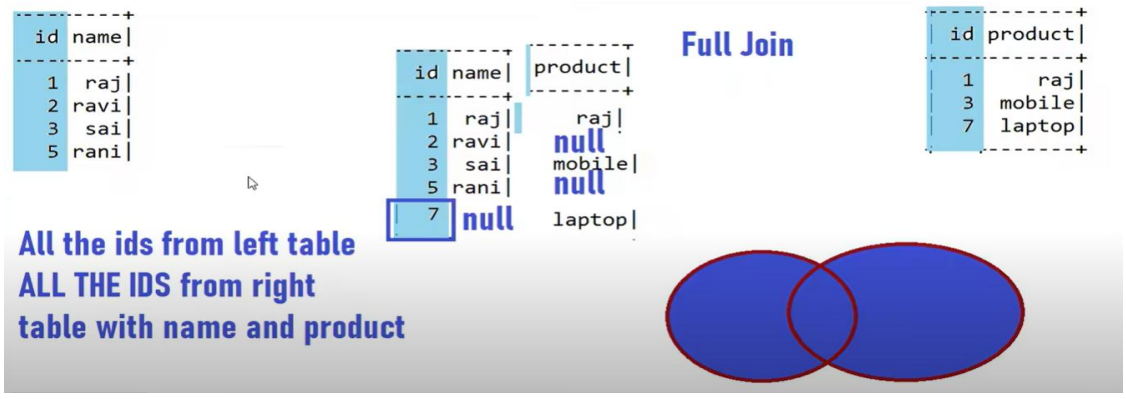
spark.sql("select a.*, b.product from cust a right join prod b on a.id=b.id ").show()

```
+----+----+-------+
|  id|name|product|
+----+----+-------+
|NULL|NULL| laptop|
|   1| raj|  mouse|
|   3| sai| mobile|
+----+----+-------+
```

==========

Full Join

==========



spark.sql("select a.*, b.product from cust a full join prod b on a.id=b.id ").show()

```
+----+----+-------+
|  id|name|product|
+----+----+-------+
|   1| raj|  mouse|
|   2|ravi|   NULL|
|   3| sai| mobile|
|   5|rani|   NULL|
|NULL|NULL| laptop|
+----+----+-------+
```

==========

Anti join

Left anti join

==========

id	name
1	raj
2	ravi
3	sai
5	rani

id	name
2	ravi
5	rani

id	product
1	raj
3	mobile
7	laptop

spark.sql("select a.* from cust a left anti join prod b on a.id=b.id ").show()

id	name
5	rani
2	ravi