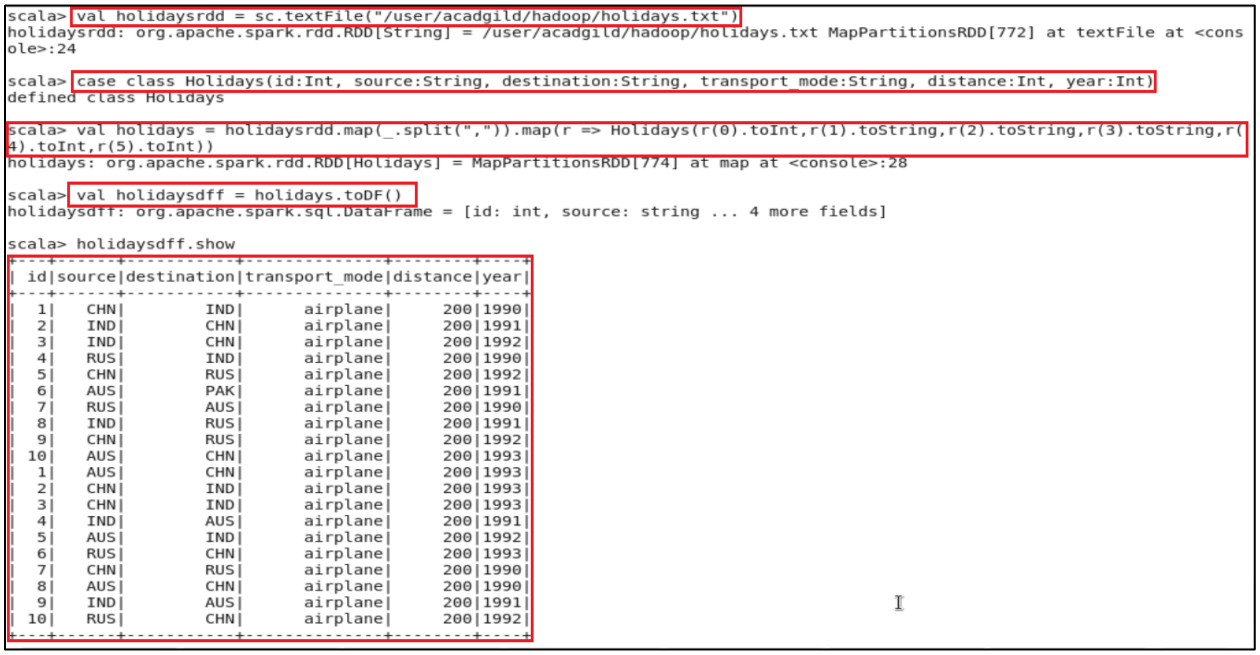
Session 20 – Spark SQL - I

Assignment 1

All the associated data files are copied to HDFS and loaded as RDDs. A data frame is created in each RDD.

Copying onto HDFS:

hadoop fs -put User_details.txt '/user/acadgild/hadoop/'
hadoop fs -put Transport.txt '/user/acadgild/hadoop/'
hadoop fs -put holidays.txt '/user/acadgild/hadoop/'



```
scala> val holidaysrdd = sc.textFile("/user/acadgild/hadoop/holidays.txt")
holidaysrdd: org.apache.spark.rdd.RDD[String] = /user/acadgild/hadoop/holidays.txt MapPartitionsRDD[772] at textFile at <cons
ole>:24

scala> case class Holidays(id:Int, source:String, destination:String, transport_mode:String, distance:Int, year:Int)
defined class Holidays

scala> val holidays = holidaysrdd.map(_.split(",")).map(r => Holidays(r(0).toInt,r(1).toString,r(2).toString,r(3).toString,r(
4).toInt,r(5).toInt))
holidays: org.apache.spark.rdd.RDD[Holidays] = MapPartitionsRDD[774] at map at <console>:28

scala> val holidaysdff = holidays.toDF()
holidaysdff: org.apache.spark.sql.DataFrame = [id: int, source: string ... 4 more fields]

scala> holidaysdff.show
+---+------+-----------+--------------+--------+----+
| id|source|destination|transport_mode|distance|year|
+---+------+-----------+--------------+--------+----+
|  1|   CHN|        IND|      airplane|     200|1990|
|  2|   IND|        CHN|      airplane|     200|1991|
|  3|   IND|        CHN|      airplane|     200|1992|
|  4|   RUS|        IND|      airplane|     200|1990|
|  5|   CHN|        RUS|      airplane|     200|1992|
|  6|   AUS|        PAK|      airplane|     200|1991|
|  7|   RUS|        AUS|      airplane|     200|1990|
|  8|   IND|        RUS|      airplane|     200|1991|
|  9|   CHN|        RUS|      airplane|     200|1992|
| 10|   AUS|        CHN|      airplane|     200|1993|
|  1|   AUS|        CHN|      airplane|     200|1993|
|  2|   CHN|        IND|      airplane|     200|1993|
|  3|   CHN|        IND|      airplane|     200|1993|
|  4|   IND|        AUS|      airplane|     200|1991|
|  5|   AUS|        IND|      airplane|     200|1992|
|  6|   RUS|        CHN|      airplane|     200|1993|
|  7|   CHN|        RUS|      airplane|     200|1990|
|  8|   AUS|        CHN|      airplane|     200|1990|
|  9|   IND|        AUS|      airplane|     200|1991|
| 10|   RUS|        CHN|      airplane|     200|1992|
+---+------+-----------+--------------+--------+----+
```

An rdd is created from the text file from HDFS.

val holidaysrdd = sc.textFile("/user/acadgild/hadoop/holidays.txt")

For specifying the schema of the dataframe, create a case class.

case class Holidays(id:Int, source:String, destination:String, transport_mode:String, distance:Int, year:Int)

Loading the values of the RDD as components of the class type Holidays.

val holidays = holidaysrdd.map(_.split(",")).map(r => Holidays(r(0).toInt,r(1).toString,r(2).toString,r(3).toString,r(4).toInt,r(5).toInt))

Creating a dataframe out of the RDD.

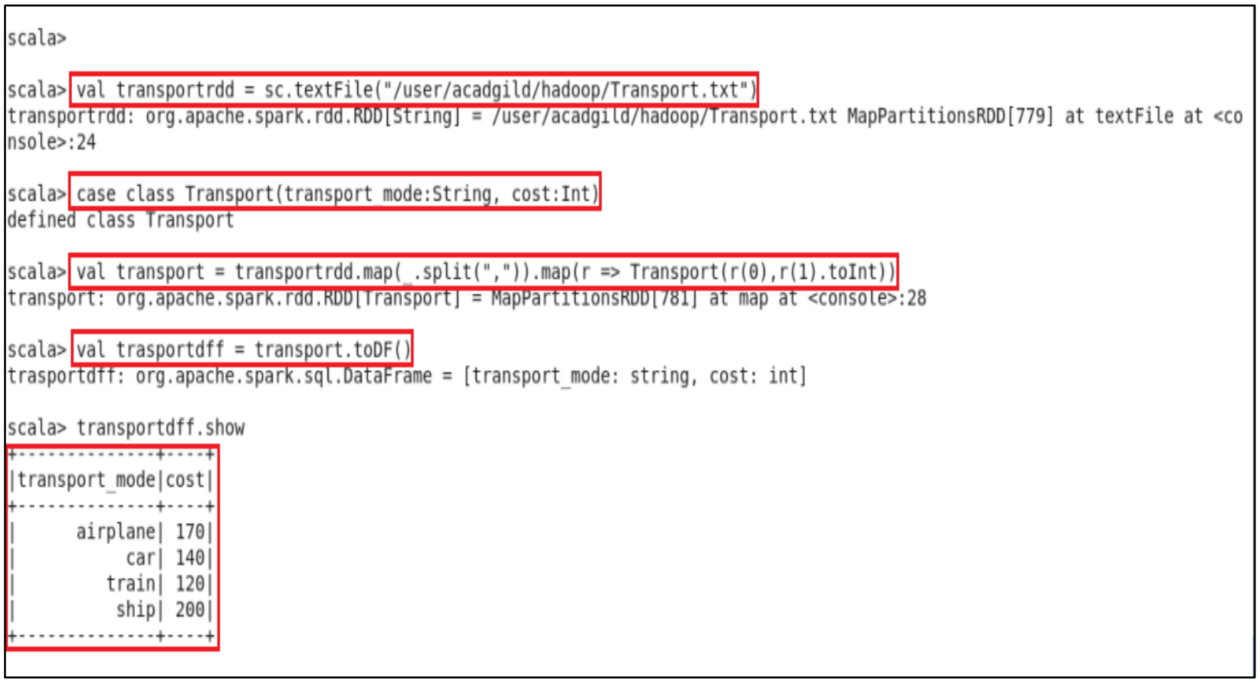
val holidaysdff = holidays.toDF()

Displaying the contents of dataframe using the following command.

holidaysdff.show

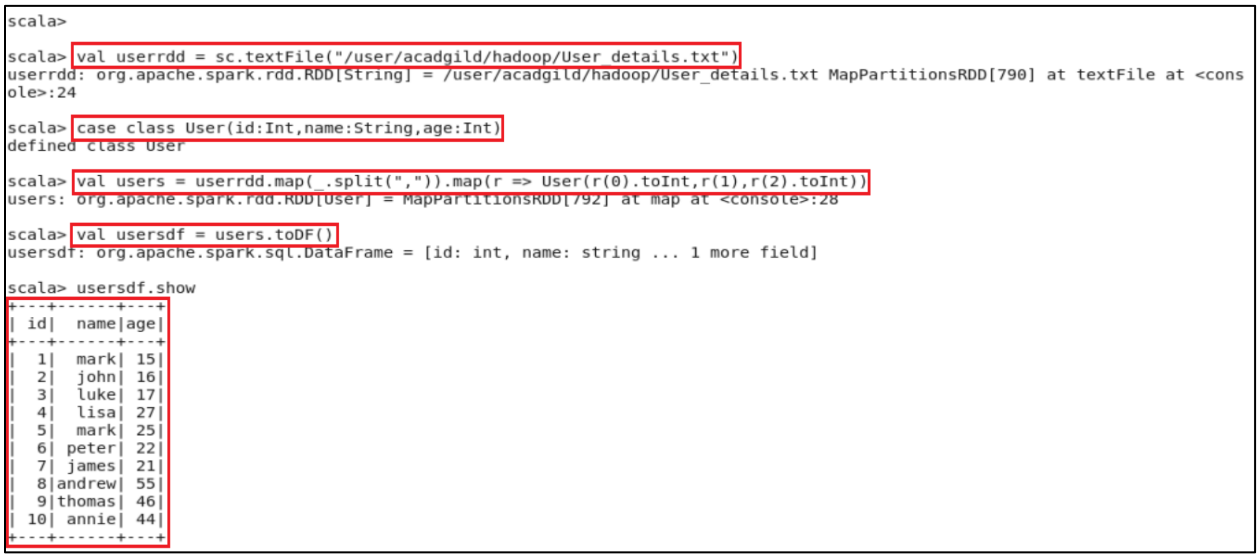
The contents of the dataframe are shown as a table.

The same process is repeated for the other 2 text files: transport.txt and User_details.txt



```
scala>

scala> val transportrdd = sc.textFile("/user/acadgild/hadoop/Transport.txt")
transportrdd: org.apache.spark.rdd.RDD[String] = /user/acadgild/hadoop/Transport.txt MapPartitionsRDD[779] at textFile at <co
nsole>:24

scala> case class Transport(transport_mode:String, cost:Int)
defined class Transport

scala> val transport = transportrdd.map(_.split(",")).map(r => Transport(r(0),r(1).toInt))
transport: org.apache.spark.rdd.RDD[Transport] = MapPartitionsRDD[781] at map at <console>:28

scala> val trasportdff = transport.toDF()
trasportdff: org.apache.spark.sql.DataFrame = [transport_mode: string, cost: int]

scala> transportdff.show
+--------------+----+
|transport_mode|cost|
+--------------+----+
|      airplane| 170|
|           car| 140|
|         train| 120|
|          ship| 200|
+--------------+----+
```

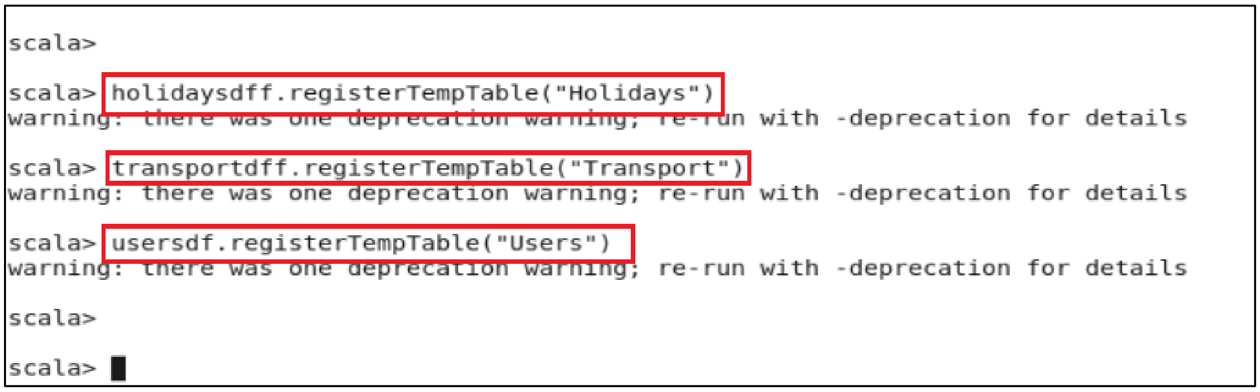
val transportrdd = sc.textFile("/user/acadgild/hadoop/Transport.txt")
case class Transport(transport_mode:String, cost:Int)
val transport = transportrdd.map(_.split(",")).map(r => Transport(r(0),r(1).toInt))
val trasportdff = transport.toDF()
transportdff.show



```
scala>

scala> val userrdd = sc.textFile("/user/acadgild/hadoop/User_details.txt")
userrdd: org.apache.spark.rdd.RDD[String] = /user/acadgild/hadoop/User_details.txt MapPartitionsRDD[790] at textFile at <cons
ole>:24

scala> case class User(id:Int,name:String,age:Int)
defined class User

scala> val users = userrdd.map(_.split(",")).map(r => User(r(0).toInt,r(1),r(2).toInt))
users: org.apache.spark.rdd.RDD[User] = MapPartitionsRDD[792] at map at <console>:28

scala> val usersdf = users.toDF()
usersdf: org.apache.spark.sql.DataFrame = [id: int, name: string ... 1 more field]

scala> usersdf.show
+---+------+---+
| id|  name|age|
+---+------+---+
|  1|  mark| 15|
|  2|  john| 16|
|  3|  luke| 17|
|  4|  lisa| 27|
|  5|  mark| 25|
|  6| peter| 22|
|  7| james| 21|
|  8|andrew| 55|
|  9|thomas| 46|
| 10| annie| 44|
+---+------+---+
```

val userrdd = sc.textFile("/user/acadgild/hadoop/User_details.txt")
case class User(id:Int,name:String,age:Int)
val users = userrdd.map(_.split(",")).map(r => User(r(0).toInt,r(1),r(2).toInt))
val usersdf = users.toDF()
usersdf.show

These Dataframes are saved as temporary tables for ease of querying.



```
scala>

scala> holidaysdff.registerTempTable("Holidays")
warning: there was one deprecation warning; re-run with -deprecation for details

scala> transportdff.registerTempTable("Transport")
warning: there was one deprecation warning; re-run with -deprecation for details

scala> usersdf.registerTempTable("Users")
warning: there was one deprecation warning; re-run with -deprecation for details

scala>

scala> █
```

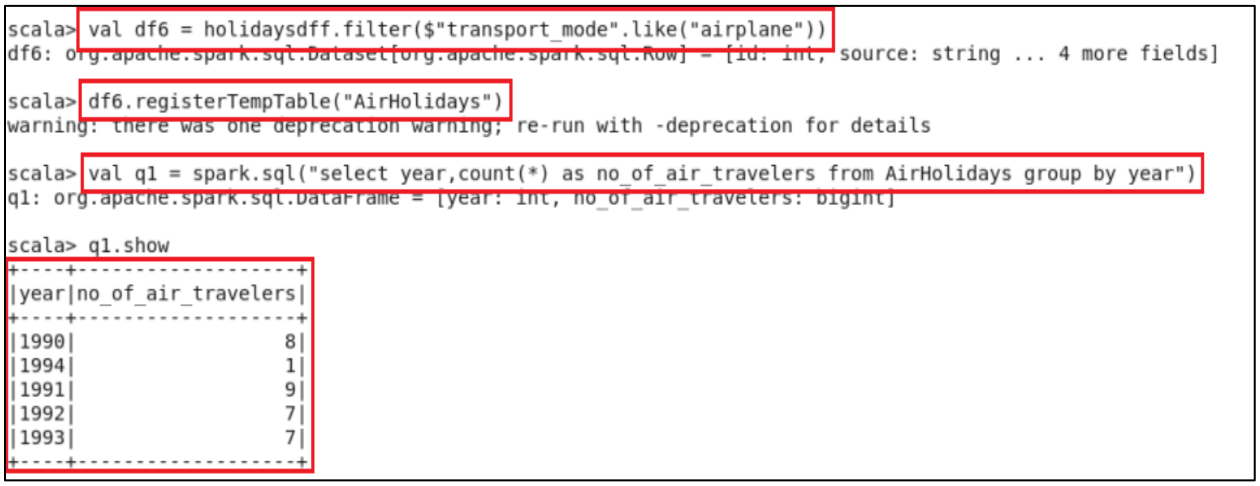
holidaysdff.registerTempTable("Holidays")

transportdff.registerTempTable("Transport")

usersdf.registerTempTable("Users")

Task 1:

1) What is the distribution of the total number of air-travelers per year?



```
scala> val df6 = holidaysdff.filter($"transport_mode".like("airplane"))
df6: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [id: int, source: string ... 4 more fields]

scala> df6.registerTempTable("AirHolidays")
warning: there was one deprecation warning; re-run with -deprecation for details

scala> val q1 = spark.sql("select year,count(*) as no_of_air_travelers from AirHolidays group by year")
q1: org.apache.spark.sql.DataFrame = [year: int, no_of_air_travelers: bigint]

scala> q1.show
+----+-------------------+
|year|no_of_air_travelers|
+----+-------------------+
|1990|                  8|
|1994|                  1|
|1991|                  9|
|1992|                  7|
|1993|                  7|
+----+-------------------+
```

We filter the holidaysdff with the transport_mode as airplane, since only the air travelers are asked.

val df6 = holidaysdff.filter($"transport_mode".like("airplane"))

Then the filtered dataset is saved as a temporary table.

df6.registerTempTable("AirHolidays")

Then, the temporary table is queried as follows:

val q1 = spark.sql("select year,count(*) as no_of_air_travelers from Holidays group by year")

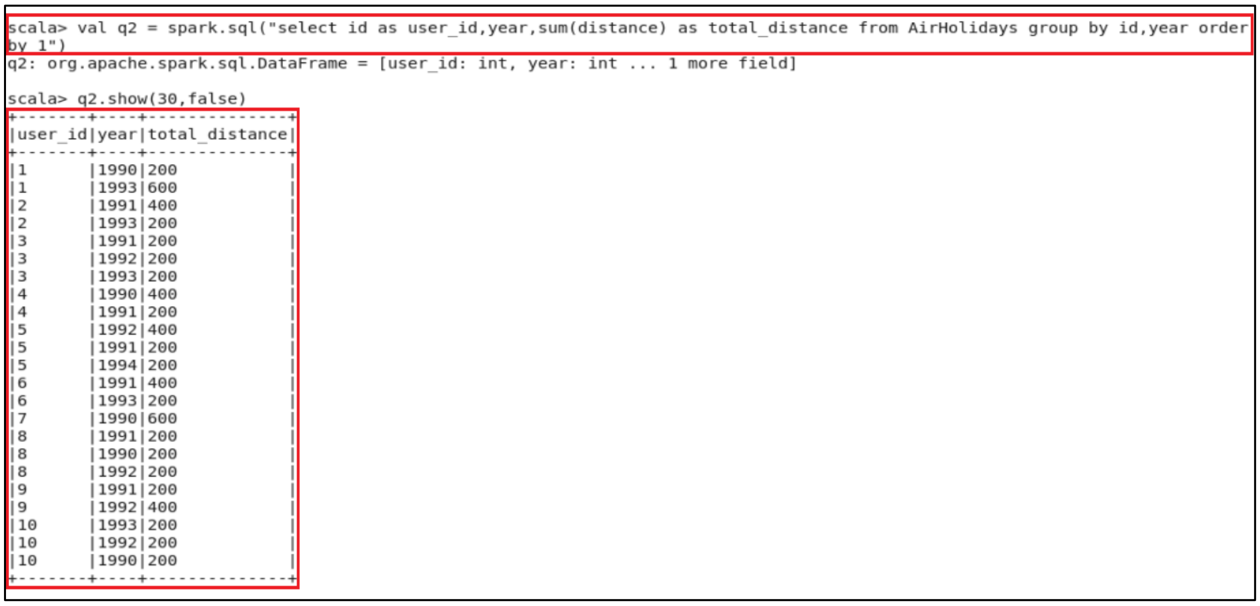
The resultant dataframe is displayed as

q1.show

The number of travelers per year are displayed in a tabular format.

2) What is the total air distance covered by each user per year?

Since only the air distance is asked, the filtered dataframe



```
scala> val q2 = spark.sql("select id as user_id,year,sum(distance) as total_distance from AirHolidays group by id,year order
by 1")
q2: org.apache.spark.sql.DataFrame = [user_id: int, year: int ... 1 more field]

scala> q2.show(30,false)
+-------+----+--------------+
|user_id|year|total_distance|
+-------+----+--------------+
|1      |1990|200           |
|1      |1993|600           |
|2      |1991|400           |
|2      |1993|200           |
|3      |1991|200           |
|3      |1992|200           |
|3      |1993|200           |
|4      |1990|400           |
|4      |1991|200           |
|5      |1992|400           |
|5      |1991|200           |
|5      |1994|200           |
|6      |1991|400           |
|6      |1993|200           |
|7      |1990|600           |
|8      |1991|200           |
|8      |1990|200           |
|8      |1992|200           |
|9      |1991|200           |
|9      |1992|400           |
|10     |1993|200           |
|10     |1992|200           |
|10     |1990|200           |
+-------+----+--------------+
```

Since the air distance is asked, we use the filtered dataset under the table name ‘AirHolidays’.

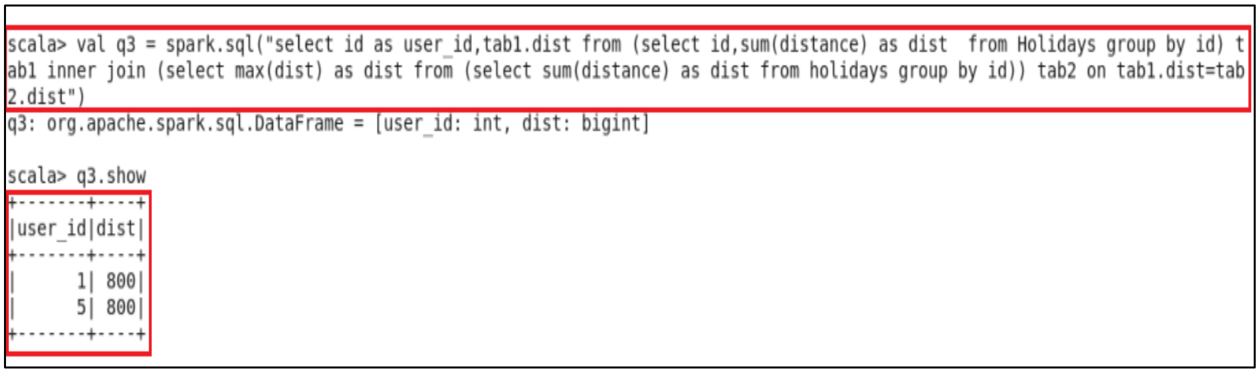
val q2 = spark.sql("select id as user_id,year,sum(distance) as total_distance from AirHolidays group by id,year")

The resultant dataframe is displayed as

q2.show

The total distance covered by each user per year is displayed.

3) Which user has travelled the largest distance till date



```
scala> val q3 = spark.sql("select id as user_id,tab1.dist from (select id,sum(distance) as dist  from Holidays group by id) t
ab1 inner join (select max(dist) as dist from (select sum(distance) as dist from holidays group by id)) tab2 on tab1.dist=tab
2.dist")
q3: org.apache.spark.sql.DataFrame = [user_id: int, dist: bigint]

scala> q3.show
+-------+----+
|user_id|dist|
+-------+----+
|      1| 800|
|      5| 800|
+-------+----+
```

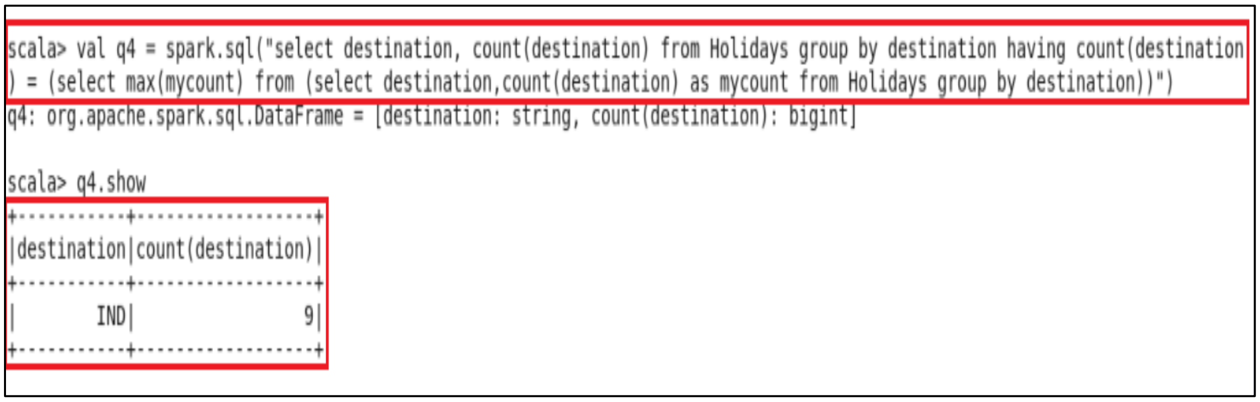
Query the temporary table Holidays as follows:

val q3 = spark.sql("select id as user_id,tab1.dist from (select id,sum(distance) as dist from Holidays group by id) tab1 inner join (select max(dist) as dist from (select sum(distance) as dist from holidays group by id)) tab2 on tab1.dist=tab2.dist")

The resultant dataframe is displayed as

q3.show

4) What is the most preferred destination for all users



```
scala> val q4 = spark.sql("select destination, count(destination) from Holidays group by destination having count(destination
) = (select max(mycount) from (select destination,count(destination) as mycount from Holidays group by destination))")
q4: org.apache.spark.sql.DataFrame = [destination: string, count(destination): bigint]

scala> q4.show
+-----------+------------------+
|destination|count(destination)|
+-----------+------------------+
|        IND|                 9|
+-----------+------------------+
```

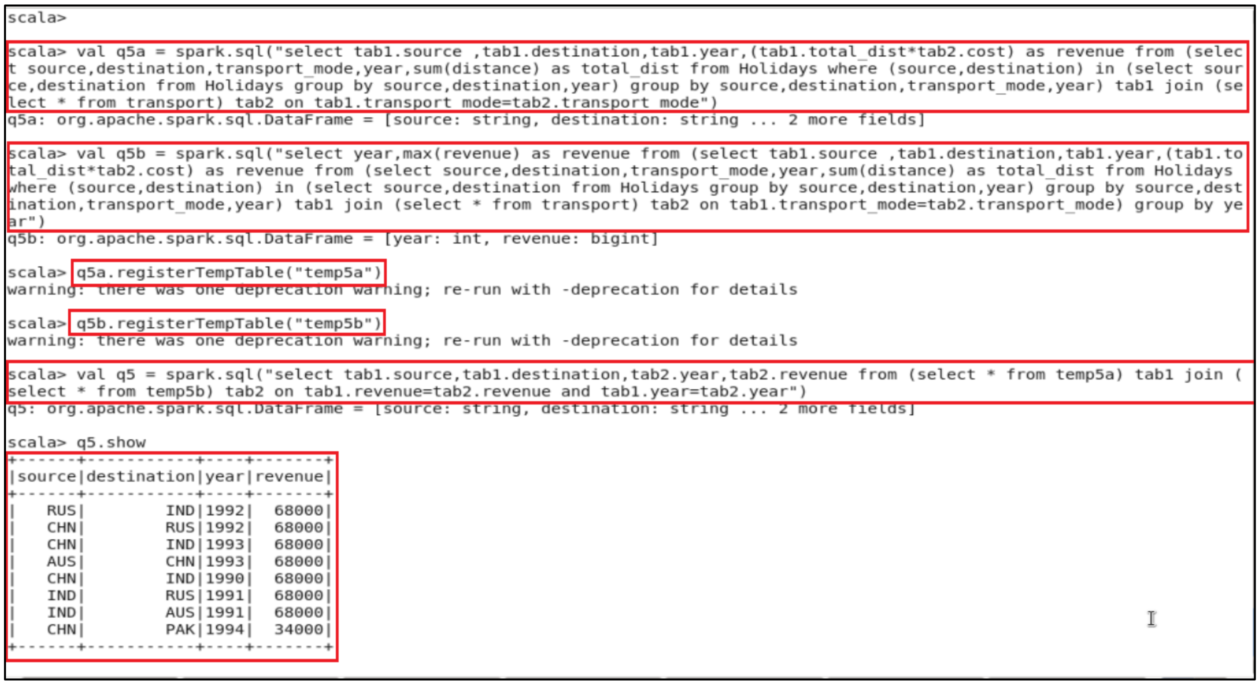
val q4 = spark.sql("select destination, count(destination) from Holidays group by destination having count(destination) = (select max(mycount) from (select destination,count(destination) as mycount from Holidays group by destination))")

The resultant dataframe is displayed as

q4.show

Since the destination IND occurs max number of times in the dataset that is displayed as the most preferred destination for all users.

5) Which route is generating the most revenue per year



```
scala>

scala> val q5a = spark.sql("select tab1.source ,tab1.destination,tab1.year,(tab1.total_dist*tab2.cost) as revenue from (selec
t source,destination,transport_mode,year,sum(distance) as total_dist from Holidays where (source,destination) in (select sour
ce,destination from Holidays group by source,destination,year) group by source,destination,transport_mode,year) tab1 join (se
lect * from transport) tab2 on tab1.transport_mode=tab2.transport_mode")
q5a: org.apache.spark.sql.DataFrame = [source: string, destination: string ... 2 more fields]

scala> val q5b = spark.sql("select year,max(revenue) as revenue from (select tab1.source ,tab1.destination,tab1.year,(tab1.to
tal_dist*tab2.cost) as revenue from (select source,destination,transport_mode,year,sum(distance) as total_dist from Holidays
where (source,destination) in (select source,destination from Holidays group by source,destination,year) group by source,dest
ination,transport_mode,year) tab1 join (select * from transport) tab2 on tab1.transport_mode=tab2.transport_mode) group by ye
ar")
q5b: org.apache.spark.sql.DataFrame = [year: int, revenue: bigint]

scala> q5a.registerTempTable("temp5a")
warning: there was one deprecation warning; re-run with -deprecation for details

scala> q5b.registerTempTable("temp5b")
warning: there was one deprecation warning; re-run with -deprecation for details

scala> val q5 = spark.sql("select tab1.source,tab1.destination,tab2.year,tab2.revenue from (select * from temp5a) tab1 join (
select * from temp5b) tab2 on tab1.revenue=tab2.revenue and tab1.year=tab2.year")
q5: org.apache.spark.sql.DataFrame = [source: string, destination: string ... 2 more fields]

scala> q5.show
+------+-----------+----+-------+
|source|destination|year|revenue|
+------+-----------+----+-------+
|   RUS|        IND|1992|  68000|
|   CHN|        RUS|1992|  68000|
|   CHN|        IND|1993|  68000|
|   AUS|        CHN|1993|  68000|
|   CHN|        IND|1990|  68000|
|   IND|        RUS|1991|  68000|
|   IND|        AUS|1991|  68000|
|   CHN|        PAK|1994|  34000|
+------+-----------+----+-------+
```

The source, destination, the corresponding year and the total revenue of each route per year is queried.

> *val q5a = spark.sql("select tab1.source ,tab1.destination,tab1.year,(tab1.total_dist*tab2.cost) as revenue from*
>
> *(select source,destination,transport_mode,year,sum(distance) as total_dist from Holidays where (source,destination) in (select source,destination from Holidays group by source,destination,year) group by source,destination,transport_mode,year) tab1*
>
> *join*
>
> *(select * from transport) tab2 on tab1.transport_mode=tab2.transport_mode")*

These results are stored in a temporary table.

> *q5a.registerTempTable("temp5a")*

The year and maximum value of the revenue for the year is queried.

> *val q5b = spark.sql("select year,max(revenue) as revenue from*
>
> *(select tab1.source ,tab1.destination,tab1.year,(tab1.total_dist*tab2.cost) as revenue from (select source,destination,transport_mode,year,sum(distance) as total_dist from Holidays where (source,destination) in (select source,destination*

from Holidays group by source,destination,year) group by source,destination,transport_mode,year) tab1

join

*(select * from transport) tab2 on tab1.transport_mode=tab2.transport_mode) group by year")*

These results are stored in a temporary table.

q5b.registerTempTable("temp5b")

The two temporary tables are joined and the final results are displayed.

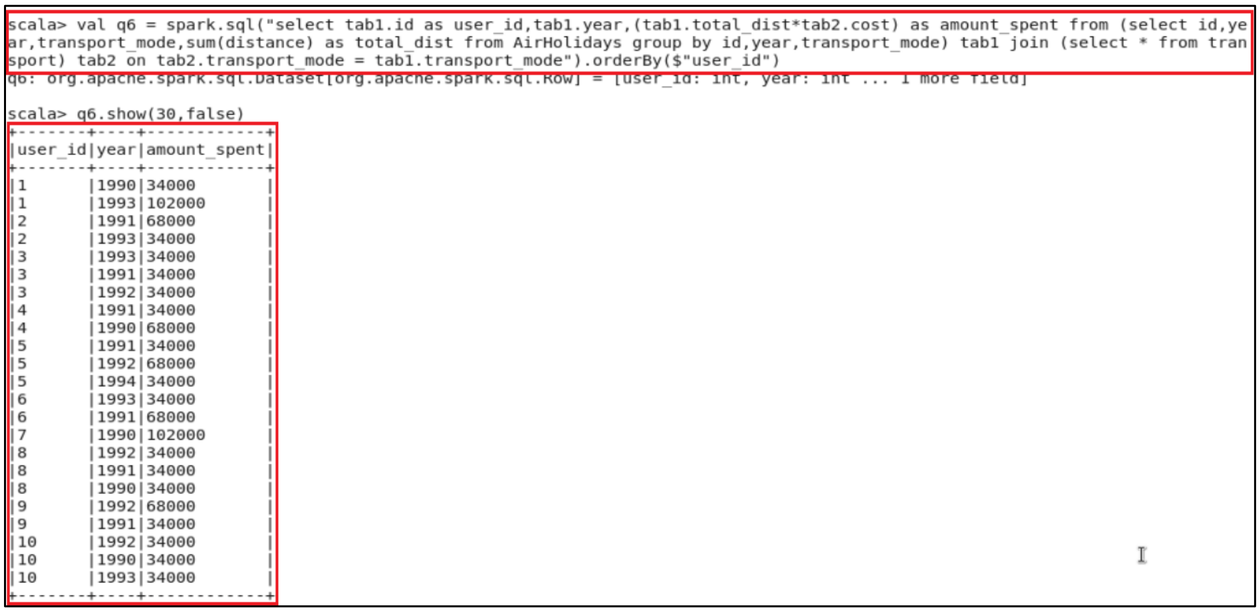
val q5 = spark.sql("select tab1.source,tab1.destination,tab2.year,tab2.revenue from
*(select * from temp5a) tab1*
join
*(select * from temp5b) tab2*
on tab1.revenue=tab2.revenue
and tab1.year=tab2.year")

The resultant dataframe is displayed as

q5.show

The source and destination specify the route and, in each year, the route that collected the highest revenue per year are displayed in the results.

6) What is the total amount spent by every user on air-travel per year



```
scala> val q6 = spark.sql("select tab1.id as user_id,tab1.year,(tab1.total_dist*tab2.cost) as amount_spent from (select id,ye
ar,transport_mode,sum(distance) as total_dist from AirHolidays group by id,year,transport_mode) tab1 join (select * from tran
sport) tab2 on tab2.transport_mode = tab1.transport_mode").orderBy($"user_id")
q6: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [user_id: int, year: int ... 1 more field]

scala> q6.show(30,false)
+-------+----+------------+
|user_id|year|amount_spent|
+-------+----+------------+
|1      |1990|34000       |
|1      |1993|102000      |
|2      |1991|68000       |
|2      |1993|34000       |
|3      |1993|34000       |
|3      |1991|34000       |
|3      |1992|34000       |
|4      |1991|34000       |
|4      |1990|68000       |
|5      |1991|34000       |
|5      |1992|68000       |
|5      |1994|34000       |
|6      |1993|34000       |
|6      |1991|68000       |
|7      |1990|102000      |
|8      |1992|34000       |
|8      |1991|34000       |
|8      |1990|34000       |
|9      |1992|68000       |
|9      |1991|34000       |
|10     |1992|34000       |
|10     |1990|34000       |
|10     |1993|34000       |
+-------+----+------------+
```

We consider the filtered dataset AirHolidays, since only the air-travel metrics are asked.

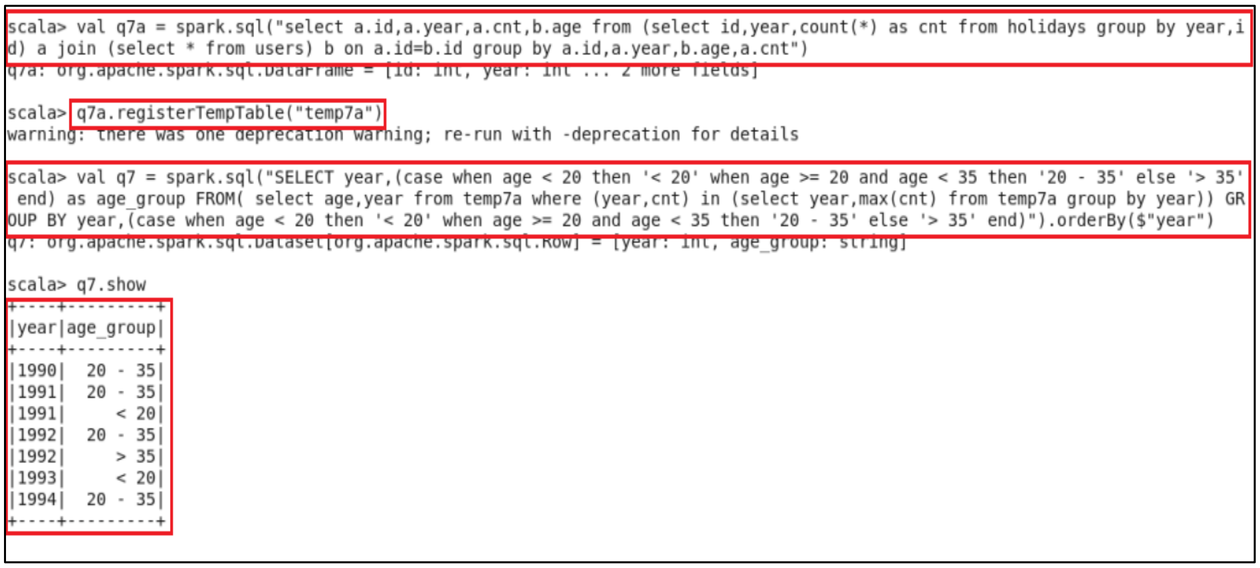
> *val q6 = spark.sql("select tab1.id as user_id,tab1.year,(tab1.total_dist*tab2.cost) as amount_spent from*
> *(select id,year,transport_mode,sum(distance) as total_dist from AirHolidays group by id,year,transport_mode) tab1*
> *join*
> *(select * from transport) tab2*
> *on*
> *tab2.transport_mode = tab1.transport_mode").orderBy($"user_id")*

The resultant dataframe is displayed as

> *q6.show*

The user_id, and the amount spent by the user year wise is displayed in the results.

7) Considering age groups of < 20 , 20-35, 35 > ,Which age group is travelling the most every year.



```
scala> val q7a = spark.sql("select a.id,a.year,a.cnt,b.age from (select id,year,count(*) as cnt from holidays group by year,i
d) a join (select * from users) b on a.id=b.id group by a.id,a.year,b.age,a.cnt")
q7a: org.apache.spark.sql.DataFrame = [id: int, year: int ... 2 more fields]

scala> q7a.registerTempTable("temp7a")
warning: there was one deprecation warning; re-run with -deprecation for details

scala> val q7 = spark.sql("SELECT year,(case when age < 20 then '< 20' when age >= 20 and age < 35 then '20 - 35' else '> 35'
 end) as age_group FROM( select age,year from temp7a where (year,cnt) in (select year,max(cnt) from temp7a group by year)) GR
OUP BY year,(case when age < 20 then '< 20' when age >= 20 and age < 35 then '20 - 35' else '> 35' end)").orderBy($"year")
q7: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [year: int, age_group: string]

scala> q7.show
+----+---------+
|year|age_group|
+----+---------+
|1990|  20 - 35|
|1991|  20 - 35|
|1991|     < 20|
|1992|  20 - 35|
|1992|     > 35|
|1993|     < 20|
|1994|  20 - 35|
+----+---------+
```

The user_id, the count of the user occurs year-wise and the user's age is queried.

> *val q7a = spark.sql("select a.id,a.year,a.cnt,b.age from*
> *(select id,year,count(*) as cnt from holidays group by year,id) a*
> *join*
> *(select * from users) b*
> *on a.id=b.id*
> *group by a.id,a.year,b.age,a.cnt")*

This result is stored as a temporary table since, this table will be used for self joining.

```
q7a.registerTempTable("temp7a")
```

The temporary table is joined to itself and case statement is used to differentiate the age groups.

```
val q7 = spark.sql("select year,(case when age < 20 then '< 20'
                                      when age >= 20 and age < 35 then '20 - 35'
                                      else '> 35'
                                      end) as age_group
FROM( select age,year from temp7a where (year,cnt) in (select year,max(cnt)
from temp7a group by year))
GROUP BY year,(case when age < 20 then '< 20'
        when age >= 20 and age < 35 then '20 - 35'
        else '> 35' end)")
.orderBy($"year")
```

The resultant dataframe is displayed as

```
q6.show
```

The year and the age group occurred the most times occurred in that year and displayed.