Problem Statement

1. Join of two or more data sets is one of the most widely used operations you do with your data, but in distributed systems it can be a huge headache. In general, since your data are distributed among many nodes, they have to be shuffled before a join that causes significant network I/O and slow performance.
2. Fortunately, if you need to join a large table with relatively small tables you can avoid sending all data of the large table over the network. This type of join is called map-side join in Hadoop community. **In other distributed systems, it is often called replicated or broadcast join.**

The fact table can be very large, while dimension tables are often quite small.

Let's use the following sample data (one fact and two dimension tables):

```
// Fact table
val flights = sc.parallelize(List(
("SEA", "JFK", "DL", "418", "7:00"),
("SFO", "LAX", "AA", "1250", "7:05"),
("SFO", "JFK", "VX", "12", "7:05"),
("JFK", "LAX", "DL", "424", "7:10"),
("LAX", "SEA", "DL", "5737", "7:10")))

// Dimension table
val airports = sc.parallelize(List(
("JFK", "John F. Kennedy International Airport", "New York", "NY"),
("LAX", "Los Angeles International Airport", "Los Angeles", "CA"),
("SEA", "Seattle-Tacoma International Airport", "Seattle", "WA"),
("SFO", "San Francisco International Airport", "San Francisco", "CA")))

// Dimension table
val airlines = sc.parallelize(List(
("AA", "American Airlines"),
("DL", "Delta Airlines"),
("VX", "Virgin America")))
```

We need to join the fact and dimension tables to get the following result:

```
Seattle New York Delta Airlines 418 7:00
San Francisco Los Angeles American Airlines 1250 7:05
San Francisco New York Virgin America 12 7:05
New York Los Angeles Delta Airlines 424 7:10
Los Angeles Seattle Delta Airlines 5737 7:10
```

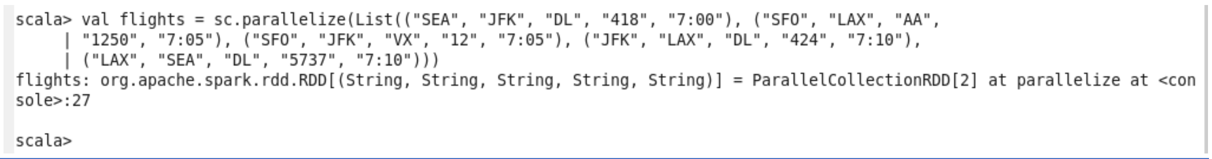
Solution:

To solve the given problem, follow the below steps:

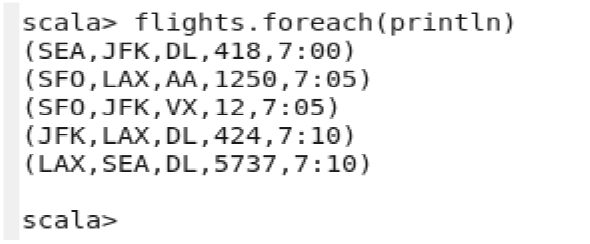
- Begin by reading the data files as text files from the local FS using the spark context object ***sc***. *flights, airports, airlines.*
- Now that the data has been loaded into respective RDDs,
  - ✓ Create a broadcast variables (shared, read-only) for both the dimension table variables,
  - ✓ Select only the 1st and 3rd column from airports and collect/transform result to a Map. airportsMap
  - ✓ Collect/Transform the data in airlines to a Map. airlinesMap
- Now that both the dimension variables have been transformed as desired and broadcasted, use the elements with the data from the fact table flights as:
  - ✓ 1st & 2nd value from the airportsMap,
  - ✓ 3rd value from airlinesMap *and*
  - ✓ 4th & 5th from flights.

This is the map-side join.

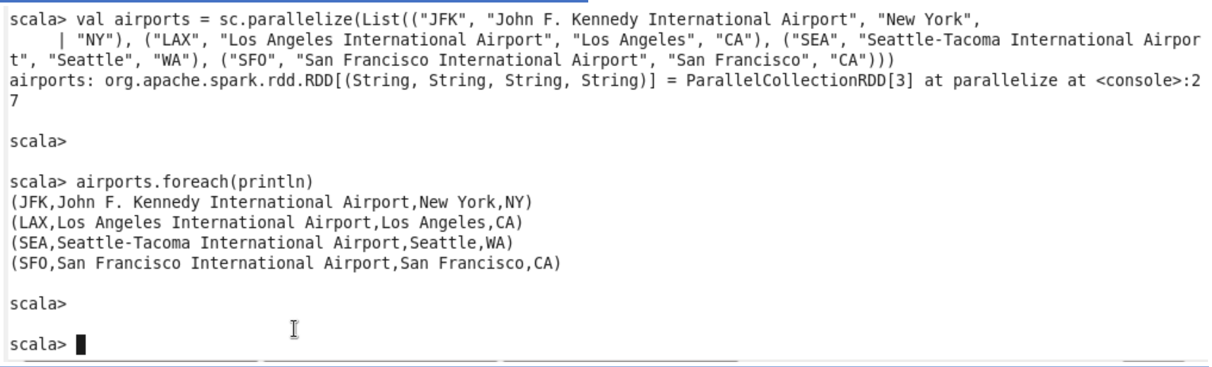
Code:



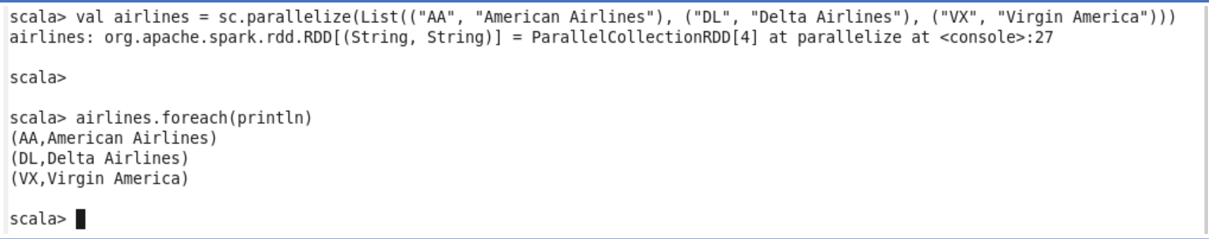
```
scala> val flights = sc.parallelize(List(("SEA", "JFK", "DL", "418", "7:00"), ("SFO", "LAX", "AA",
     | "1250", "7:05"), ("SFO", "JFK", "VX", "12", "7:05"), ("JFK", "LAX", "DL", "424", "7:10"),
     | ("LAX", "SEA", "DL", "5737", "7:10")))
flights: org.apache.spark.rdd.RDD[(String, String, String, String, String)] = ParallelCollectionRDD[2] at parallelize at <con
sole>:27

scala>
```



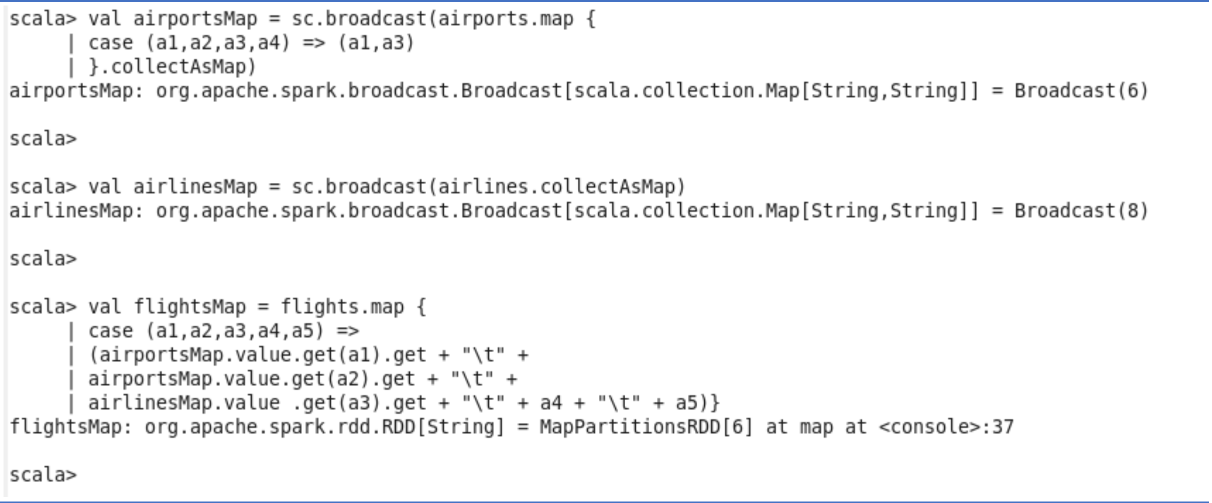
```
scala> flights.foreach(println)
(SEA,JFK,DL,418,7:00)
(SFO,LAX,AA,1250,7:05)
(SFO,JFK,VX,12,7:05)
(JFK,LAX,DL,424,7:10)
(LAX,SEA,DL,5737,7:10)

scala>
```



```
scala> val airports = sc.parallelize(List(("JFK", "John F. Kennedy International Airport", "New York",
     | "NY"), ("LAX", "Los Angeles International Airport", "Los Angeles", "CA"), ("SEA", "Seattle-Tacoma International Airpor
t", "Seattle", "WA"), ("SFO", "San Francisco International Airport", "San Francisco", "CA")))
airports: org.apache.spark.rdd.RDD[(String, String, String, String)] = ParallelCollectionRDD[3] at parallelize at <console>:2
7

scala>

scala> airports.foreach(println)
(JFK,John F. Kennedy International Airport,New York,NY)
(LAX,Los Angeles International Airport,Los Angeles,CA)
(SEA,Seattle-Tacoma International Airport,Seattle,WA)
(SFO,San Francisco International Airport,San Francisco,CA)

scala>

scala>
```

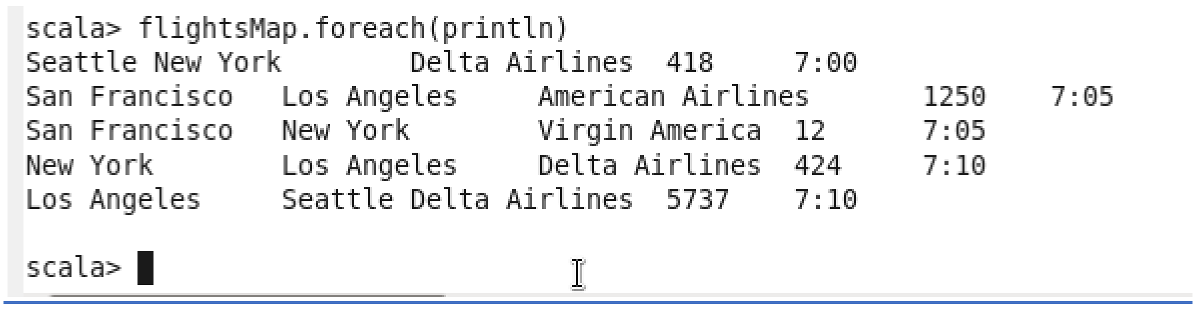


```
scala> val airlines = sc.parallelize(List(("AA", "American Airlines"), ("DL", "Delta Airlines"), ("VX", "Virgin America")))
airlines: org.apache.spark.rdd.RDD[(String, String)] = ParallelCollectionRDD[4] at parallelize at <console>:27

scala>

scala> airlines.foreach(println)
(AA,American Airlines)
(DL,Delta Airlines)
(VX,Virgin America)

scala>
```



```
scala> val airportsMap = sc.broadcast(airports.map {
     | case (a1,a2,a3,a4) => (a1,a3)
     | }.collectAsMap)
airportsMap: org.apache.spark.broadcast.Broadcast[scala.collection.Map[String,String]] = Broadcast(6)

scala>

scala> val airlinesMap = sc.broadcast(airlines.collectAsMap)
airlinesMap: org.apache.spark.broadcast.Broadcast[scala.collection.Map[String,String]] = Broadcast(8)

scala>

scala> val flightsMap = flights.map {
     | case (a1,a2,a3,a4,a5) =>
     | (airportsMap.value.get(a1).get + "\t" +
     | airportsMap.value.get(a2).get + "\t" +
     | airlinesMap.value .get(a3).get + "\t" + a4 + "\t" + a5)}
flightsMap: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[6] at map at <console>:37

scala>
```

Output:



```
scala> flightsMap.foreach(println)
Seattle New York        Delta Airlines  418     7:00
San Francisco   Los Angeles     American Airlines       1250    7:05
San Francisco   New York        Virgin America  12      7:05
New York        Los Angeles     Delta Airlines  424     7:10
Los Angeles     Seattle Delta Airlines  5737    7:10

scala> 
```