Below we have constants set up to help with readability.

```
In [1]: val hotelCountryId = 21

        hotelCountryId = 21

Out[1]: 21
```

```
In [2]: val hotelMarketId = 22

        hotelMarketId = 22

Out[2]: 22
```

```
In [3]: val hotelContinentId = 20

        hotelContinentId = 20

Out[3]: 20
```

```
In [4]: val srchAdultsCntId = 15

        srchAdultsCntId = 15

Out[4]: 15
```

This is a helper lambda that extracts only the necessary fields from our csv file.

We will be grouping by:

- hotel country
- hotel market
- hotel continent

And sorting by:

- search adults count.

In [5]:
```
val extractNecessaryFields : String => (String, String, String, Int) = (line: String) => {
    val splitLine = line.split(",")
    val hotelCountry = splitLine(hotelCountryId)
    val hotelMarket = splitLine(hotelMarketId)
    val hotelContinent = splitLine(hotelContinentId)
    val srchAdultsCnt = splitLine(srchAdultsCntId)
    val srchAdultsCntInt = Integer.parseInt(srchAdultsCnt)
    (hotelCountry, hotelMarket, hotelContinent, srchAdultsCntInt)
}
```

```
extractNecessaryFields = > (String, String, String, Int) = <function1>
```

Out[5]:
```
> (String, String, String, Int) = <function1>
```

Below we set up the config and the context of our application.

In [6]:
```
import org.apache.spark.SparkConf
val sparkConf = new SparkConf().setAppName("task1").setMaster("local")
```

```
sparkConf = org.apache.spark.SparkConf@5a7240de
```

Out[6]:
```
org.apache.spark.SparkConf@5a7240de
```

In [14]:
```
import org.apache.spark.SparkContext
val sc = new SparkContext(sparkConf)
```

```
sc = org.apache.spark.SparkContext@39ba04fe

lastException: Throwable = null
```

Out[14]:
```
org.apache.spark.SparkContext@39ba04fe
```

Here I create an RDD the train.csv file.

In [15]:
```
val data = sc.textFile("train.csv")
```

```
data = train.csv MapPartitionsRDD[1] at textFile at <console>:32
```

Out[15]:
```
train.csv MapPartitionsRDD[1] at textFile at <console>:32
```

The main function performs the following steps:

- Skips the header
- Extracts the fields that we need to use in our query
- Filters out non-couples
- Groups everything by hotel country, hotel market, hotel continent
- Sorts everything by the number of group repetitions in descending order
- Leaves only top 3 results
- Prints everything to the screen

In [26]:
```
val header = data.first() // header
val result = data.filter(row => row != header) // skip header
    .map(extractNecessaryFields) // extract necessary fields
    .filter(_._4 == 2) // only choose couples
    .groupBy(row => (row._1, row._2, row._3)) // group by hotel country, hotel
 market and hotel continent
    .mapValues(_.size) // transform Iterable[(String,String,String,Int)] into
 Iterable[Int]
    .sortBy(kv => kv._2, false) // sort by the number of people in descending
 order
    .take(3) // leave only top 3 results
```

```
header = date_time,site_name,posa_continent,user_location_country,user_locati
on_region,user_location_city,orig_destination_distance,user_id,is_mobile,is_p
ackage,channel,srch_ci,srch_co,srch_adults_cnt,srch_children_cnt,srch_rm_cnt,
srch_destination_id,srch_destination_type_id,is_booking,cnt,hotel_continent,h
otel_country,hotel_market,hotel_cluster
result = Array(((50,628,2),127031), ((50,675,2),92729), ((8,110,4),55163))
```

Out[26]:
```
Array(((50,628,2),127031), ((50,675,2),92729), ((8,110,4),55163))
```

In [11]:
```
sc.stop()
```

```
lastException: Throwable = null
```