StackOverflow

```
// Input file
z.put("stackoverflow", "/home/ubuntu/test_spark/stackoverflow/stackoverflow.csv")
```

```
%sh
head {stackoverflow}
```

```
1,27233496,,,0,C#
1,23698767,,,9,C#
1,5484340,,,0,C#
2,5494879,,5484340,1,
1,9419744,,,2,Objective-C
1,26875732,,,1,C#
1,9002525,,,2,C++
2,9003401,,9002525,4,
2,9003942,,9002525,1,
2,9005311,,9002525,0,
```

```
/** A raw stackoverflow posting, either a question or an answer */

type QID = Int
type HighScore = Int
type LangIndex = Int

case class Posting(postingType: Int, id: Int, acceptedAnswer: Option[Int], parentId: Option[QID], score: Int, tags: Option[String]) extends Serializable

type Question = Posting
type Answer = Posting
```

```
defined type alias QID
defined type alias HighScore
defined type alias LangIndex
defined class Posting
defined type alias Question
defined type alias Answer
```

```
// Start reading and transforming csv file

val lines = sc.textFile(z.get("stackoverflow").toString)

lines.take(5)
```

```
lines: org.apache.spark.rdd.RDD[String] = /home/ubuntu/test_spark/stackoverflow/stackoverflow.csv MapPartitionsRDD[697] at textFile at <console>:169
res159: Array[String] = Array(1,27233496,,,0,C#, 1,23698767,,,9,C#, 1,5484340,,,0,C#, 2,5494879,,5484340,1,, 1,9419744,,,2,Objective-C)
```

```
/** Load postings from the given file */

/**
 * <postTypeId>,<id>,[<acceptedAnswer>],[<parentId>],<score>,[<tag>]
 *
 * A short explanation of the comma-separated fields follows.
 * <postTypeId>:     Type of the post. Type 1 = question, type 2 = answer.
 * <id>:             Unique id of the post (regardless of type).
 * <acceptedAnswer>: Id of the accepted answer post. This information is optional, so maybe be missingindicated by an empty string.
 * <parentId>:       For an answer: id of the corresponding question. For a question:missing, indicated by an empty string.
 * <score>:          The StackOverflow score (based on user votes).
 * <tag>:            The tag indicates the programming language that the post is about, in case it's a question, or missing in case it's an answer.
**/


import org.apache.spark.rdd.RDD

def rawPostings(lines: RDD[String]): RDD[Posting] = {
    lines.map(line => {
        val elems = line.split(",")
        Posting(postingType = elems(0).toInt,
                id = elems(1).toInt,
                acceptedAnswer = if (elems(2) == "") None else Some(elems(2).toInt),
                parentId = if (elems(3) == "") None else Some(elems(3).toInt),
                score = elems(4).toInt,
                tags = if (elems.length >= 6) Some(elems(5).intern()) else None)
    })
}
```

```
import org.apache.spark.rdd.RDD
rawPostings: (lines: org.apache.spark.rdd.RDD[String])org.apache.spark.rdd.RDD[Posting]
```

```
val raw = rawPostings(lines)
raw.take(5)
```

```
raw: org.apache.spark.rdd.RDD[Posting] = MapPartitionsRDD[698] at map at <console>:185
res160: Array[Posting] = Array(Posting(1,27233496,None,None,0,Some(C#)), Posting(1,23698767,None,None,9,Some(C#)), Posting(1,5484340,None,None,0,Some(C#)), Posting(2,5494879,None,Some(5484340),1,None), P
```

```
import org.apache.spark.HashPartitioner
val q = raw
        .filter(_.postingType == 1)
        .map(question => (question.id, question))
        .partitionBy(new HashPartitioner(100))
q.take(5)
```

```
import org.apache.spark.HashPartitioner
q: org.apache.spark.rdd.RDD[(Int, Posting)] = ShuffledRDD[701] at partitionBy at <console>:170
res161: Array[(Int, Posting)] = Array((3406000,Posting(1,3406000,None,None,0,Some(C#))), (17520100,Posting(1,17520100,None,None,1,Some(JavaScript))), (23620700,Posting(1,23620700,None,None,0,Some(PHP))),
```

```
q.partitions.length
```

```
res162: Int = 100
```

```
// ** repartition is not an action **

val q1 = q.repartition(10)
```

```
q1: org.apache.spark.rdd.RDD[(Int, Posting)] = MapPartitionsRDD[705] at repartition at <console>:165
```

```
val answers = raw
              .filter(_.postingType == 2)
              .map(answer => (answer.parentId.get, answer))
              .partitionBy(new HashPartitioner(100))
answers.take(5)
```

```
answers: org.apache.spark.rdd.RDD[(QID, Posting)] = ShuffledRDD[644] at partitionBy at <console>:154
res142: Array[(QID, Posting)] = Array((3406000,Posting(2,3406081,None,Some(3406000),0,None)), (3406000,Posting(2,3408739,None,Some(3406000),0,None)), (17933700,Posting(2,17935422,None,Some(17933700),5,No
```

```
answers.partitions.length
```

```
res84: Int = 100
```

```
val joined = q.join(answers)
joined.take(5)
```

```
joined: org.apache.spark.rdd.RDD[(Int, (Posting, Posting))] = MapPartitionsRDD[647] at join at <console>:153
res143: Array[(Int, (Posting, Posting))] = Array((7885200,(Posting(1,7885200,None,None,0,Some(JavaScript)),Posting(2,7885231,None,Some(7885200),2,None))), (637100,(Posting(1,637100,None,None,8,Some(Java)
```

```
val groupedPostings = joined.groupByKey()
groupedPostings.take(5)
```

```
groupedPostings: org.apache.spark.rdd.RDD[(Int, Iterable[(Posting, Posting)])] = MapPartitionsRDD[648] at groupByKey at <console>:151
res144: Array[(Int, Iterable[(Posting, Posting)])] = Array((7885200,CompactBuffer((Posting(1,7885200,None,None,0,Some(JavaScript)),Posting(2,7885231,None,Some(7885200),2,None)))), (20746400,CompactBuffer
```

```
/** Group the questions and answers together */

def groupedPostings(postings: RDD[Posting]): RDD[(QID, Iterable[(Question, Answer)])] = {
    // Create questions rdd (id, question)
    val questions = postings
                    .filter(_.postingType == 1)
                    .map(question => (question.id, question))
                    .partitionBy(new HashPartitioner(100))
    // Create answers rdd (parentId, answer)
    val answers = postings
                   .filter(_.postingType == 2)
                   .map(answer => (answer.parentId.get, answer))
                   .partitionBy(new HashPartitioner(100))

    // 1 question will have multiple answers
    // Join both the rdds (inner join)
    val joined = questions.join(answers)

    // Finally group the results
    joined.groupByKey()
}
```

```
groupedPostings: (postings: org.apache.spark.rdd.RDD[Posting])org.apache.spark.rdd.RDD[(QID, Iterable[(Question, Answer)])]
```

```
val grouped = groupedPostings(raw)
grouped.take(5)
```

```
grouped: org.apache.spark.rdd.RDD[(QID, Iterable[(Question, Answer)])] = MapPartitionsRDD[658] at groupByKey at <console>:179
res145: Array[(QID, Iterable[(Question, Answer)])] = Array((7885200,CompactBuffer((Posting(1,7885200,None,None,0,Some(JavaScript)),Posting(2,7885231,None,Some(7885200),2,None)))), (20746400,CompactBuffer
```

```
// Compute score

// val scored: RDD[(Question, HighScore)] = ???

val score = grouped.map{case (qid, qu_ans) => {
    // val question = qu_ans.head match {
    //     case (q: Question, ans: Answer) => q
    // }
    //(question, 1)

    val scores = qu_ans.map{case (q, ans) => ans.score}

    (qu_ans.head._1, scores, scores.max)
}}

score.take(5)
```

```
score: org.apache.spark.rdd.RDD[(Question, Iterable[Int], Int)] = MapPartitionsRDD[659] at map at <console>:155
res146: Array[(Question, Iterable[Int], Int)] = Array((Posting(1,7885200,None,None,0,Some(JavaScript)),List(2),2), (Posting(1,20746400,None,None,1,Some(C#)),List(2, 2, 1),2), (Posting(1,637100,None,None,
```

```
/** Compute the maximum score for each posting */
def scoredPostings(grouped: RDD[(QID, Iterable[(Question, Answer)])]): RDD[(Question, HighScore)] = {
    val score = grouped.map{case(qid, qu_ans) => {
        (qu_ans.head._1, qu_ans.map{case(q, ans) => ans.score}.max)
    }}
    score
}
```

```
scoredPostings: (grouped: org.apache.spark.rdd.RDD[(QID, Iterable[(Question, Answer)])])org.apache.spark.rdd.RDD[(Question, HighScore)]
```

```
val scored = scoredPostings(grouped)
scored.take(5)
```

```
scored: org.apache.spark.rdd.RDD[(Question, HighScore)] = MapPartitionsRDD[660] at map at <console>:160
res147: Array[(Question, HighScore)] = Array((Posting(1,7885200,None,None,0,Some(JavaScript)),2), (Posting(1,20746400,None,None,1,Some(C#)),2), (Posting(1,637100,None,None,8,Some(Java)),8), (Posting(1,15
```

```
/** Languages */
val langs = List(
    "JavaScript", "Java", "PHP", "Python", "C#", "C++", "Ruby", "CSS",
    "Objective-C", "Perl", "Scala", "Haskell", "MATLAB", "Clojure", "Groovy")

/** Compute the vectors for the kmeans */
def vectorPostings(scored: RDD[(Question, HighScore)]): RDD[(LangIndex, HighScore)] = {
    val vector = scored.map{case(q, highScore) => {
        val tag = q.tags.getOrElse("")
        val index = langs.indexOf(tag)
        (index * 50000, highScore)
    }}
    vector
}
```

```
langs: List[String] = List(JavaScript, Java, PHP, Python, C#, C++, Ruby, CSS, Objective-C, Perl, Scala, Haskell, MATLAB, Clojure, Groovy)
vectorPostings: (scored: org.apache.spark.rdd.RDD[(Question, HighScore)])org.apache.spark.rdd.RDD[(LangIndex, HighScore)]
```

```
val vectors = vectorPostings(scored)
vectors.take(5)
```

```
vectors: org.apache.spark.rdd.RDD[(LangIndex, HighScore)] = MapPartitionsRDD[661] at map at <console>:164
res148: Array[(LangIndex, HighScore)] = Array((0,2), (200000,2), (50000,8), (50000,2), (50000,3))
```

```
assert(vectors.count() == 2121822, "Incorrect number of vectors: " + vectors.count())
```

```
vectors.count
```

```
res157: Long = 2121822
```

```
vectors.par
```