Lezione 1:

Scala (Scalable Language) è un linguaggio di programmazione java-oriented/object-oriented e multi-paradigma (richiede una JVM), ed è un linguaggio molto usato in ambito big data.

La JVM (Java Virtual Machine) è un ambiente di esecuzione che è possibile aggiungere in un qualsiasi sistema operativo

L'ambiente REPL (Read-Eval-Print-Loop) è un tool scala analogo all'interprete command-line di molti linguaggi.

Utile per test e verifica di porzioni di codice, ed è di due tipi: riga comando e online.

Scastie è un playground interattivo collegato a REPL che ci permette di simulare qualsiasi versione di Scala.

Scala presenta diversi operatori, abbiamo quelli di Confronto, ovvero:

- **== (uguale a);**
- **!= (diverso da);**
- **> (maggiore di);**
- **< (minore di);**
- **>= (maggiore o uguale a);**
- **<= (minore o uguale a).**

Abbiamo quelli Logici, ovvero:

- **&& (AND);**
- **|| (OR);**

- ! (NOT).

Abbiamo quelli Aritmetici, ovvero:

- + (somma);
- - (sottrazione);
- * (moltiplicazione);
- / (divisione);
- % (resto della divisione);
- ** (esponenziazione).

E infine abbiamo quelli di Incremento, ovvero:

- ++ (incremento di una variabile di 1);
- -- (decremento di una variabile di 1).

Scala ha due tipi di variabili, ovvero val e var.

Val rappresenta una variabile immutabile, il cui valore non può essere modificato dopo l'assegnazione iniziale.

Var rappresenta una variabile mutabile, il cui valore può essere modificato durante il ciclo di vita del programma.

Le Funzioni Anonime sono funzioni in scala senza nome, e sono definite dal simbolo "=>" che separa la lista degli argomenti della funzione dal suo corpo

Tutte le Collection class si trovano nel package "scala.collection" e possono essere mutabili e immutabili.

Esempi di collection:

- Map ("x" -> 24, "y" -> 25, "z" -> 26);

- **LinearSeq (a, b, c);**
- **IndexedSeq (1.0, 2.0);**
- **Set (color.red, color.green, color.blue);**
- **SortedSet (“Hello”, “World”);**
- **Buffer (x, y, z);**
- **Trasversable(1, 2, 3);**
- **Iterable (“x”, “y”, “z”).**

Codice riga di comando:

```
Microsoft Windows [Versione 10.0.19045.2965]
(c) Microsoft Corporation. Tutti i diritti sono riservati.

C:\WINDOWS\system32>scala
Welcome to Scala 3.3.0 (1.8.0_372-372, Java OpenJDK 64-Bit Server VM).
Type in expressions for evaluation. Or try :help.

scala>

scala> :help
The REPL has several commands available:

:help                    print this summary
:load <path>             interpret lines in a file
:quit                    exit the interpreter
:type <expression>       evaluate the type of the given expression
:doc <expression>        print the documentation for the given expression
:imports                 show import history
:reset [options]         reset the repl to its initial state, forgetting all session entries
:settings <options>      update compiler options, if possible


scala> var mutabile = 10
var mutabile: Int = 10

scala> mutabile = 11
mutabile: Int = 11

scala> val b: Byte = 1
val b: Byte = 1

scala> val l: Long = 1
val l: Long = 1

scala> val speech = "Four score and seven years ago our fathers"
val speech: String = Four score and seven years ago our fathers

scala> println(speech)
Four score and seven years ago our fathers

scala> speech.charAt(3)
val res0: Char = r

scala> val contenitore = (3, 5, 3, 8, 7)
val contenitore: (Int, Int, Int, Int, Int) = (3,5,3,8,7)

scala> var somma = 0
var somma: Int = 0

scala> println(contenitore._1)
3

scala> somma = (contenitore._1) + (contenitore._3)
somma: Int = 6

scala> println("La somma è " + somma)
La somma è 5

scala> val frase = "Con affetto e simpatia, buongiorno pescheria"
val frase: String = Con affetto e simpatia, buongiorno pescheria

scala> frase.substring(4, 10)
val res1: String = affett
```

```
scala> frase.drop(39)
val res2: String = heria

scala> frase.replace("pescheria", "giacomo")
val res3: String = Con affetto e simpatia, buongiorno giacomo

scala> val a = 10
val a: Int = 10

scala> val b = 20
val b: Int = 20

scala> if(a < b) a else b
val res4: Int = 10

scala> val seq: Seq[Int] = Seq(4, 23, 5, 12, 8)
val seq: Seq[Int] = List(4, 23, 5, 12, 8)

scala> seq.sorted
val res5: Seq[Int] = List(4, 5, 8, 12, 23)

scala> seq.appended(12)
val res6: Seq[Int] = List(4, 23, 5, 12, 8, 12)

scala> seq.toList
val res7: List[Int] = List(4, 23, 5, 12, 8)

scala> val ritornaValore: Int => String = { z =>
     | if(z == 10) ("Ottimo! valore " + z)
     | else if(z > 30) "Maggiore della media"
     | else("Non corretto!!!")
     | }
val ritornaValore: Int => String = Lambda$1608/1346532545@16e5c19f

scala> val immutabile = 10
val immutabile: Int = 10

scala> immutabile = 11
-- [E052] Type Error: -----------------------------------------------------------
1 |immutabile = 11
  |^^^^^^^^^^^^^^^
  |Reassignment to val immutabile
  |
  | longer explanation available when compiling with `-explain`
1 error found

scala>
```

Lezione 2:

L'Interpreter Java fa parte della JVM e interpreta i file di classe Java.

Decodifica ogni bytecode ed esegue l'operazione corrispondente.

La List è una collection immutabile, dove ogni volta che si vuole creare o modificare una lista sarà necessario crearne una nuova.

L'Array è un tipo speciale di collection.

Il tipo dell'array è generico, cioè puoi avere un array[T], dove T type parameter or abstract type.

Supporta tutte le sequence operations.

L'array non è una sequenza, ma può essere wrappata come tale in quanto esiste una classe ArrayWrapper che si è fatto è una sottoclasse di Seq.

Map è un iterable, consistente di coppia chiave-valore.

Le operazioni che si possono fare sono simili a quelle del costrutto Set.

Le Eccezioni sono eventi anomali o situazioni impreviste che si verificano durante l'esecuzione di un programma.

Un'eccezione può essere gestita da vari fattori, come: errori di sintassi, errore di accesso ai dati, problemi di connettività di rete o errori di calcolo.

Scala ha un solo vero statement di loop: While.

While essendo non-functional viene spesso rimpiazzato da function map, anche se sulle collections è preferibile utilizzare il For.

ReadLine -> Leggere la riga di comando;

PrintWriter -> Scrivere su un file.

Una classe in Scala è una definizione di un tipo di oggetto che può avere proprietà, metodi e costruttori può essere istanziata per creare oggetti multipli.

Le classi possono avere campi mutabili o immutabili, a seconda delle dichiarazioni dei campi come var o val.

Le case class sono una forma speciale di classe utilizzata principalmente per modellare dati immutabili.

Utilizzate per rappresentare strutture dati complesse e sono ottimizzate per confrontare strutture e pattern matching.

Forniscono alcune funzionalità extra automaticamente generate come metodi equals, hashcode, toString e copy, che sono basati sui valori dei campi e non sull'identità dell'oggetto.

La classe astratta è una classe che non può essere istanziata.

In Scala, l'espressione Match è simile all'espressione switch di Java.

Match opera, a seconda del valore da confrontare, scegliendo l'operazione che deve processare.

Non è limitato in valori interi, ma può utilizzare qualsiasi tipo di dato.

Esempio di take e drop:

val hello = "Hello World"

hello.take(5) -> prende le prime 5 lettere in considerazione (Hello);

hello.drop(5) -> toglie le prime 5 lettere in considerazione(World).

Printf è una funzione che permette di formattare una stringa utilizzando specificatori di formato come %s, %d, %f, etc.

Puoi utilizzare printf per definire un modello di formattazione e sostituire i segnaposto con i valori delle variabili o espressioni utilizzando gli argomenti successivi

Println con l'interpolazione di stringhe stampa espressioni o variabili separate dalle virgole, covertendole in stringhe.

Puoi inserire i valori delle variabili direttamente nella stringa utilizzando la sintassi $variabile o ${espressione}.

```
1  var numero = 10
2  printf("numero: %d", numero)  (): scala.Unit
3
4
```

```
numero: 10
```

```scala
var numero = 10
println(s"numero: $numero") ": scala.Unit
```

```
numero: 10
```

Codice riga di comando:

```
Microsoft Windows [Versione 10.0.19045.2965]
(c) Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\utente>scala
Welcome to Scala 3.3.0 (1.8.0_372-372, Java OpenJDK 64-Bit Server VM).
Type in expressions for evaluation. Or try :help.

scala> val lista = List(1, 2, 3)
val lista: List[Int] = List(1, 2, 3)

scala> val lista1: List[Int] = List(1, 2, 3, 4)
val lista1: List[Int] = List(1, 2, 3, 4)

scala> val listaStringa: List[String] = List("oggi", "domani", "dopo domani")
val listaStringa: List[String] = List(oggi, domani, dopo domani)

scala> val array = Array(1, 2, 3)
val array: Array[Int] = Array(1, 2, 3)

scala> array(0)
val res0: Int = 1

scala> array(1)
val res1: Int = 2

scala> array(2)
val res2: Int = 3

scala> array(5)
java.lang.ArrayIndexOutOfBoundsException: 5
  ... 41 elided

scala> val array1 = array.map(_ * 3)
val array1: Array[Int] = Array(3, 6, 9)

scala> val array2 = array1.filter(_ % 2 !=0)
val array2: Array[Int] = Array(3, 9)

scala> array.reverse
val res4: Array[Int] = Array(3, 2, 1)

scala> val seq1 = array1
val seq1: Array[Int] = Array(3, 6, 9)

scala> val seq2: Seq[Int] = array1
there was 1 deprecation warning; re-run with -deprecation for details
1 warning found
val seq2: Seq[Int] = ArraySeq(3, 6, 9)

scala> seq1 eq seq2
val res5: Boolean = false

scala> val posata = Map("forchetta" -> "fork",
     | "cucchiaio" -> "spoon",
     | "coltello" -> "knife")
val posata: Map[String, String] = Map(forchetta -> fork, cucchiaio -> spoon, coltello -> knife)

scala> posata("cucchiaio")
val res6: String = spoon

scala> posata("forchetta")
val res7: String = fork
```

```
scala> posata("cucchiaino")
java.util.NoSuchElementException: key not found: cucchiaino
  at scala.collection.immutable.Map$Map3.apply(Map.scala:397)
  ... 41 elided

scala> val divided = try {
     | 10/0
     | } catch {
     | case ae: ArithmeticException => 0
     | } finally {
     | println("Errore risultato")}
Errore risultato
val divided: Int = 0

scala> 10/0
java.lang.ArithmeticException: / by zero
  ... 41 elided

scala> val numers = Seq(1, 2, 3, 4, 5)
val numers: Seq[Int] = List(1, 2, 3, 4, 5)

scala> for(num <- numers) println(num)
1
2
3
4
5

scala> var interi = List(10, 20, 30, 40, 50)
var interi: List[Int] = List(10, 20, 30, 40, 50)

scala> for(i <- interi) {
     | println(s"numero: $i")
     | }
numero: 10
numero: 20
numero: 30
numero: 40
numero: 50

scala> val ints = List(1, 2, 3, 4, 5)
val ints: List[Int] = List(1, 2, 3, 4, 5)

scala> val doubles = for(i <- ints) yield i * 2
val doubles: List[Int] = List(2, 4, 6, 8, 10)

scala> val i = 1
val i: Int = 1

scala> var i = 0
var i: Int = 0
```

```
scala> while(i < 10) {
     | val quadrato = i * i
     | println(s"il quadrato di $i è $quadrato")
     | i = i + 1
     | }
il quadrato di 0 è 0
il quadrato di 1 è 1
il quadrato di 2 è 4
il quadrato di 3 è 9
il quadrato di 4 è 16
il quadrato di 5 è 25
il quadrato di 6 è 36
il quadrato di 7 è 49
il quadrato di 8 è 64
il quadrato di 9 è 81

scala> val i = 1
val i: Int = 1

scala> i match {
     | case 1 => println("uno")
     | case 2 => println("due")
     | case _ => println("unico")
     | }
uno

scala> val i = 4
val i: Int = 4

scala> i match {
     | case 1 => println("uno")
     | case 2 => println("due")
     | case _ => println("unico")
     | }
unico

scala>
```

Lezione 3:

Le classi possono ereditare attributi e metodi usando la key-word extends.

Spark è un framework di elaborazione distribuita e calcolo parallelo progettato per affrontare problemi di analisi e elaborazione di dati su larga scala.

Lo SparkContext è il punto di entrata principale per le funzionalità di spark e rappresenta la connessione al cluster e può essere usato per creare gli RDD.

Il processo driver è un processo che esegue la funzione main(), coordinando gli executors;

Il processo executor risiede nei nodi worker, ed elabora le richieste del driver.

Il Cluster Manager è un software che gestisce le risorse del cluster e coordina l'esecuzione delle applicazioni su di esso.

Il RDD (Resilient Distributed Datasets) è una raccolta di elementi fault-tolerant su cui si può operare in parallelo.

E' possibile creare un RDD con la parallelizzazione della raccolta oppure facendo riferimento a un set di dati in un sistema di archiviazione esterno, come l'HDFS.

Codice riga di comando:

```
scala> val nomiCognomiMinuscolo = List("mario rossi", "giulia bianchi", "paolo verdi")
val nomiCognomiMinuscolo: List[String] = List(mario rossi, giulia bianchi, paolo verdi)

scala> def convertiInMaiuscolo(nomeCognome: String): String ={
     | val parti = nomeCognome.split(" ")
     | val nomeMaiuscolo = parti(0).toUpperCase
     | val cognomeMaiuscolo = parti(1).toUpperCase
     | s"$nomeMaiuscolo $cognomeMaiuscolo"}
def convertiInMaiuscolo(nomeCognome: String): String
```

```
scala> val nomiCognomiMaiuscolo = nomiCognomiMinuscolo.map(convertiInMaiuscolo)
val nomiCognomiMaiuscolo: List[String] = List(MARIO ROSSI, GIULIA BIANCHI, PAOLO VERDI)

scala>
```

Lezione 4:

Una trasformazione è ogni operazione di Spark che restituisce un RDD (quando richiamata da un Dataframe o un Dataset restituisce la stessa tipologia di struttura dati).

Le trasformazioni sono eseguite in modo lazy.

Esempi di trasformazione sono map(func), filter(func), flatMap(func), mapPartitions(func), etc…

Map() è un'operazione che viene utilizzata per applicare la trasformazione a ogni elemento di RDD.

Essendo una trasformazione, avrà come risultato un nuovo RDD.

La trasformazione Spark flatMap() appiattisce la colonna RDD dopo aver applicato la funzione su ogni elemento e restituisce rispettivamente un nuovo RDD.

La trasformazione filter() permette di filtrare un RDD definendo una condizione tramite lambda expression.

Trattandosi di una trasformazione, il risultato sarà un nuovo RDD, potenzialmente con meno elementi.

La trasformazione join() permette di joinare due RDD definendo una condizione.

Il metodo groupBy() serve per specificare l'elemento della riga (quindi il sotto-elemento dell'RDD).

La trasformazione groupByKey() permette di poter eseguire funzioni di aggregazione.

Il metodo mean() permette di calcolare la media aritmetica devi valori contenuti in un RDD.

Le azioni, invece, non sono eseguite in modo lazy.

Quando inseriamo un'azione nel codice e Spark raggiunge quella riga di codice durante l'esecuzione del lavoro, dovrà eseguire tutte le trasformazioni che portano a quell'azione per produrre un valore.

Mentre le trasformazioni restituiscono uno dei tipi di dati di Spark, le azioni restituiscono un conteggio di elementi (ad esempio, la funzione count), un elenco di elementi (collect, take, etc...) o memorizzano i dati in un archivio esterno (write, saveAsTextFile e altre).

Come abbiamo già detto, esempio di azioni sono reduce(func), collect, count(), first(), take(n), foreach(func), etc…

Il metodo collect() è un'azione che restituisce tutti gli elementi dell'RDD Questo metodo sfrutta la memoria del driver, quindi è sconsigliato usarlo quando l'RDD è molto grande.

L'azione first() è un'azione che restituisce il primo elemento dell'RDD

Il metodo take() è un'azione che restituisce il numero di elementi, passato come parametro, dell'RDD.

Lezione 5:

Un DataFrame è un insieme immutabile di records organizzato in colonne.

Rappresenta i dati come una tabella strutturata: può leggere e scrivere dati in diversi formati strutturati (ad esempio, JSON, tabelle Hive, Parquet, Avro, ORC, CSV).

Può essere usato con SparkSQL, e fornisce il motore su cui sono costruite le API strutturate di alto livello.

Per creare un DataFrame, acquisendo i dati in lettura da un file, è necessario utilizzare l'oggetto SparkSession.

Nel metodo csv() è possibile specificare il percorso di un file o un percorso di un folder che deve contenere i diversi file da cui leggere i dati.

Il metodo show() mostra i primi elementi del DataFrame, mentre il metodo printSchema() mostra lo schema dei dati.

Per definire uno schema è necessario creare un oggetto di tipo StructType().

Il metodo Il metodo withColumn() permette di creare una nuova colonna.

Il metodo drop() permette di rimuovere una colonna.

Il Dataset è una struttura dati che ha caratteristiche in comune sia con i Dataframe che con gli RDD.

Deve essere tipizzato come gli RDD e il Dataset deve essere tipizzato come una case class.

Un Dataframe è un Dataset di Row.

Le Aggregazioni sono operazioni che consentono di calcolare valori aggregati basati su uno o più attributi del DataFrame.

Spark SQL è utilizzato per l'esecuzione di query SQL.

Spark SQL può anche essere utilizzato per leggere i dati da un'installazione Hive esistente.

Quando si esegue SQL da un altro linguaggio di programmazione, i risultati vengono restituiti come Dataset/DataFrame.

È inoltre possibile interagire con l'interfaccia SQL tramite la riga di comando o tramite JDBC/ODBC.

Le UDF (User-Defined Functions) sono una caratteristica fondamentale del framework di elaborazione dei dati Apache Spark.

Le UDF consentono agli utenti di definire funzioni personalizzate per operare su dati all'interno di Spark, e consentono anche di eseguire operazioni complesse e specifiche che non sono fornite nativamente da Spark.

Codice IntelliJ IDEA:

HelloWorld:

```
object HelloWorld
{
  def main(args: Array[String]):Unit={
    println("Hello, world!")
  }
}
```

HelloWorld2:

```
object HelloWorld2 extends App
{
  println("Hello, world!")
}
```

Calcolo modulo tre con una sequenza di numeri (filter):

```
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.log4j.{Level, Logger}

object Calcolo_modulo_3_sequenza_numeri extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  val conf = new SparkConf().setMaster("local[*]").setAppName("Test
Parallelize")
  val sc = new SparkContext(conf)

  val sequenza = Seq(25, 13, 45, 67, 31, 29, 30, 78, 21, 33)
  val operazioneRDD = sc.parallelize(sequenza)
  val operazioneRDD2 = operazioneRDD.filter(sequenze => sequenze % 3 == 0)
```

```
  operazioneRDD2.foreach(println)
}
```

DataFrameDB_Insert:

```
import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.Row
import org.apache.spark.sql.types.{IntegerType, StringType, StructType}
import org.apache.spark.sql.{SparkSession}


object DataFrameDB_Insert extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  // Senza lo SparkSession non posso creare uno spark DataFrame
  val spark = SparkSession.builder()
    .appName("DataFrameExample")
    .master("local[*]")
    .getOrCreate()

  val structureData = Seq(
    Row("James,,", "", "Smith", "36636", "NewYork", 3100),
    Row("Michael,", "Rose,", "", "40288", "California", 4300),
    Row("Robert,,", "", "Williams", "42114", "Florida", 1400),
    Row("Maria,", "Anne,", "Jones,", "39192", "Florida", 5500),
    Row("Jen,", "Mary,", "Brown", "34561", "NewYork", 3000)
  )

  val structureSchema = new StructType()
    .add("firstname", StringType)
    .add("middlename", StringType)
    .add("lastname", StringType)
    .add("id", StringType)
    .add("location", StringType)
    .add("salary", IntegerType)

  val df2 = spark.createDataFrame(
    spark.sparkContext.parallelize(structureData), structureSchema)
  df2.printSchema()
  df2.show(false)

  import spark.implicits._
  val df3 = df2.map(row => {
    val fullName = row.getString(0) + row.getString(1) + row.getString(2)
    (fullName, row.getString(3), row.getInt(5))
  }).toDF("fullName", "id", "salary")

  df3.printSchema()
  df3.show(false)
}
```

DataFrameDB_Insert2_Array:

```
import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.Row
import org.apache.spark.sql.types.{ArrayType, IntegerType, StringType,
StructType}
```

```
import org.apache.spark.sql.SparkSession

object DataFrameDB_Insert2_Array extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  // Senza lo SparkSession non posso creare uno spark DataFrame
  val spark = SparkSession.builder()
    .appName("DataFrameExample")
    .master("local[*]")
    .getOrCreate()

  val arrayStructureData = Seq(
    Row("James,,Smith", List("Java", "Scala", "C++"), "CA"),
    Row("Michael,Rose", List("Spark", "Java", "C++"), "NJ"),
    Row("Robert,,Williams", List("CSharp", "VB", "R"), "NV")
  )

  val arrayStructureSchema = new StructType()
    .add("name", StringType)
    .add("languageAtSchool", ArrayType(StringType))
    .add("currentState", StringType)

  val df = spark.createDataFrame(
    spark.sparkContext.parallelize(arrayStructureData), arrayStructureSchema)
  import spark.implicits._

  // flatMap() Usage
  val df2 = df.flatMap(f =>
f.getSeq[String](1).map((f.getString(0),_,f.getString(2))))
    .toDF("Name", "Language", "State")

  df2.show(false)
}
```

DataFrameDB_withColumn:

```
import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.Row
import org.apache.spark.sql.types.{StringType, StructType}
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.lit

object DataFrameDB_withColumn extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)
  val sparkSes = SparkSession.builder().master("local[1]")
    .appName("SparkExample").getOrCreate()

  val data = Seq(
    Row(Row("Gianpiero", " ", "Rosa"), "34", "M"),
    Row(Row("Maria", "Antonia", "Rossi"), "25", "F"),
    Row(Row("Pippo", " ", "Paperino"), "74", "M"))

  val schema = new StructType().add("name", new StructType()
    .add("PrimoNome", StringType)
    .add("MiddleName", StringType)
    .add("Surname", StringType))
    .add("eta", StringType).add("sesso", StringType)
```

```
  val dif = sparkSes.createDataFrame(sparkSes.sparkContext.parallelize(data),
schema)

  dif.show()

  //aggiunge nuova colonna chiamata
  val nuovo = dif.withColumn("country", lit("USA"))
    .withColumn("altra_colonna", lit("alta"))

  nuovo.show()
}
```

Esempio ciclo for con list:

```
object Esempio_ciclo_for_con_list  extends App
{
  val lista = List(1, 2, 3, 4, 5, 6, 7, 8, 9)

  for (num <- lista)
  {
    if (num % 2 == 0)
    {
      println(num)
    }
  }
}
```

Esempio ciclo for con seq:

```
object Esempio_ciclo_for_con_seq extends App
{
  val numeri: Seq[Int] = 1 to 10

  for(num <- numeri)
  {
    if(num % 2 == 0)
      {
        println(num)
      }
  }
}
```

Esempio DataFrame con file csv:

```
import org.apache.spark.sql.SparkSession
import org.apache.log4j.{Level, Logger}

object EsempioDataFrame extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  val sparkSes =
SparkSession.builder().master("local[*]").appName("SparkExample").getOrCreate()

  val readDF = sparkSes.read.csv("C:\\Users\\utente\\Desktop\\datafile.csv")
```

```
  readDF.show()
  readDF.printSchema()
}
```

Esempio DataFrame con file csv 2:

```
import org.apache.spark.sql.SparkSession
import org.apache.log4j.{Level, Logger}

object EsempioDataFrame2 extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  val spark =
SparkSession.builder().master("local[*]").appName("SparkExample").getOrCreate()

  val df = spark.read.option("header",
true).csv("C:\\Users\\utente\\Desktop\\datafile.csv")

  df.show()
  df.printSchema()
}
```

Esempio DataFrame con collegamento al DB:

```
import org.apache.spark.sql.SparkSession
import org.apache.log4j.{Level, Logger}
//andare su my sql installer poi connectorj upgrade
//andare su inelj file projectur structure libraries premere +
//inserire il path copiato di(questopc cerca mysql_java,cliccajava source e
copia path)
//una volta fatto cio apply e ok

object EsempioDataFrameDB extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  val sparkSes =
SparkSession.builder().master("local[1]").appName("SparkExample").getOrCreate()

  val jdbcProva = sparkSes.read.format("jdbc")
    .option("url", "jdbc:mysql://127.0.0.1:3306/ospedale_db")
    .option("dbtable", "visita")
    .option("driver", "com.mysql.cj.jdbc.Driver")
    .option("user", "root")
    .option("password", "root")
    .load()

  jdbcProva.show()
}
```

Esempio DataFrame con il Group By:

```
import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.{DataFrame, SparkSession}
import org.apache.spark.sql.functions._

object GrouppBy extends App
{

  Logger.getRootLogger.setLevel(Level.WARN)
  val spark = SparkSession.builder()
    .appName("GroupBy Example")
    .master("local")
    .getOrCreate()

  import spark.implicits._

  val simpleData = Seq(
    ("James", "Sales", "NY", 90000, 34, 10000),
    ("Michael", "Sales", "NY", 86000, 56, 20000),
    ("Robert", "Sales", "CA", 81000, 30, 23000),
    ("Maria", "Finance", "CA", 90000, 24, 23000),
    ("Raman", "Finance", "CA", 99000, 40, 24000),
    ("Scott", "Finance", "NY", 83000, 36, 19000),
    ("Jen", "Finance", "NY", 79000, 53, 15000),
    ("Jeff", "Marketing", "CA", 80000, 25, 18000),
    ("Kumar", "Marketing", "NY", 91000, 50, 21000)
  )

  val df = simpleData.toDF("employee_name", "department", "state", "salary",
"bonus", "commission")
  df.show()

  val groupedDF = df.groupBy("department", "state")
    .agg(
      sum("salary").as("total_salary"),
      avg("salary").as("average_salary")
    )
  groupedDF.show()

  val minMaxSalaryDF = df.select(min("salary").alias("min_salary"),
max("salary").alias("max_salary"))
  minMaxSalaryDF.show()
  minMaxSalaryDF.show()
}
```

Esempio con group by:

```
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.log4j.{Level, Logger}

object EsempioGroupBy extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  val conf = new SparkConf().setMaster("local[*]").setAppName("Test
Parallelize")
  val sc = new SparkContext(conf)

  val personaRDD = sc.parallelize(Array(("Tom", 16), ("Sam", 24), ("Frank", 41),
```

```
("Sam", 45), ("Frank", 21), ("Frank", 21)))

  val groupByRDD = personaRDD.groupBy(x => x._1).mapValues(_.size)

  groupByRDD.foreach(println)
}
```

Esempio join DataFrame con tabella piena:

```
import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.{Row, SparkSession, functions}
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._

object EsempioJoinDF_tabellaPiena extends App {
  Logger.getRootLogger.setLevel(Level.WARN)

  val sparkSes =
SparkSession.builder().master("local[1]").appName("SparkExample").getOrCreate()
  //mi creo prima i due seq senza row perche non li voglio in tabella
  val data = Seq( //
    ("Foggia","Giampiero", "", "Rosa", "34", "M"),
    ("Molfetta", "Haria", "Antonia", "Rossi", "25", "F"),
    ("Bari","Pippo", "", "Paperino", "74", "M")
  )

  val emp = sparkSes.createDataFrame(data).toDF("luogo_di_residenza","nome",
"secondo_nome", "cognome", "età", "sesso")
  emp.show() //qui mi creo la tabella

  val dataAddr = Seq(
    ("Bari", "78188", "via Napoli", "34"),
    ("Foggia", "75421", "via Palermo", "25"),
    ("Melfetta", "76123", "via Terlizzi", "74")
  )

  val emp2 = sparkSes.createDataFrame(dataAddr).toDF("città", "cap", "via",
"num_civico")
  emp2.show()

  val joinDF = emp.join(emp2, emp("luogo_di_residenza") === emp2("città"),
"inner") //join(unisce con altra tabella)
  //emp("nome") === emp2("cap") per questo la tabella esce vuota perche il
valore di citta non è uguale al nome
  joinDF.show()
}
```

Esempio join DataFrame con tabella vuota:

```
import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.SparkSession

object EsempioJoinDF_tabellaVuota extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  // Senza lo SparkSession non posso creare uno spark DataFrame
  val sparkSes = SparkSession.builder()
```

```
    .appName("DataFrameExample")
    .master("local[1]")
    .getOrCreate()

  val data = Seq(
    ("Giampiero", "", "Rosa", "34", "M"),
    ("Maria", "Antonia", "Rossi", "25", "F"),
    ("Pippo", "", "Paperino", "74", "M"))

  val emp = sparkSes.createDataFrame(data).toDF("nome", "secondo nome",
"cognome", "età", "sesso")
    emp.show()

  val dataAddr = Seq(
    ("Bari", "70100", "via Napoli", "34"),
    ("Foggia", "75421", "via Palermo", "25"),
    ("Molfetta", "76123", "via Terlizzi", "74"))

  val emp2 = sparkSes.createDataFrame(dataAddr).toDF("città", "cap", "via", "num
civico")
  emp2.show()

  // flatMap() Usage
  val joinDF = emp.join(emp2, emp("nome") === emp2("città"), "inner")
  joinDF.show()
}
```

Esempio Mean (calcolo della media):

```
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.log4j.{Level, Logger}

object EsempioMean extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)


  val conf = new SparkConf().setMaster("local[*]").setAppName("test")
  val sc = new SparkContext(conf)

  val personaRDD = sc.parallelize(Array(("Tom", 16),("sam", 24),("frank",41),
    ("sam",45),("frank", 21),("frank", 21)))

  val meanRDD = personaRDD.map(f => f._2).mean()
  // del secondo elemento mi fai la media

  println(meanRDD)
}
```

Esempio UDF:

```
import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.functions.udf
import org.apache.spark.sql.{DataFrame, SparkSession}

object EsempioUDF extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)
```

```
  val sparkSes = SparkSession.builder()
    .appName("GroupBy Example")
    .master("local")
    .getOrCreate()

  val data = Seq(
    ("10", "furono protagonisti di un'evoluzione incredibile delle strutture
sociali"),
    ("20", "i popoli dell'era classica trovarono il modo di raccontarsi con
poemi epici"),
    ("30", "la complessità dei popoli europei medievali, umana e sociale, trovò
espressione in molte nuove opere"),
    ("40", "queste storie, oggi, ci giungono n forme e con stili a noi poco
familiari"),
    ("50", "i contenuti di alcune sono assolutamente accattivanti"))

  val emp = sparkSes.createDataFrame(data).toDF("segno", "frase")
  emp.show(false)

  // Realizziamo una UDF che appunto faccia la trasformazione
  val convertUDF = udf ((strString: String) => {
    val indicaFrase = strString.split(" ")
    indicaFrase.map(f => f.capitalize).mkString(" ")
  })

  // Definisce e registra la UDF
  sparkSes.udf.register("converti", convertUDF)

  //Creiamo una vista ed effettuiamo la query con la trasformazione inserita
  emp.createOrReplaceTempView("nuovaFrase")
  sparkSes.sql("select segno, converti(frase) from nuovaFrase").show(false)
}
```

Inversione:

```
object Inversione extends App
{
  def invertiParole(input: String): String =
  {
    val parole = input.split(" ")
    val paroleInvertite = parole.reverse
    paroleInvertite.mkString(" ")
  }

  val inputString = "Programmazione in Scala"
  println("Prima: " + inputString)
  val outputString = invertiParole(inputString)
  println("Dopo: " + outputString)
}
```

LetturaFile:

```
import java.io.{FileOutputStream, PrintWriter}
import scala.io.Source

object LetturaFile extends App
{
```

```
  val nomeFile = "Esempio.txt"
  val variabile_path = "C:\\Users\\utente\\Desktop\\Scala\\Esempio.txt"
  val variabile_sorgente = Source.fromFile(variabile_path)

  val writer = new PrintWriter(new
FileOutputStream("C:\\Users\\utente\\Desktop\\Scala\\Esempio.txt"))
  for(line <- variabile_sorgente.getLines())
    {
      println(line)
      writer.write(line.toUpperCase() + "\n")
    }

  writer.close()
  variabile_sorgente.close()
}
```

LetturaRigaComando:

```
import scala.io.StdIn.readLine

object LetturaRigaComando extends App
{
  print("Inserisci il tuo nome: ")
  val firstName = readLine()

  print("Inserisci il tuo cognome: ")
  val lastName = readLine()

  print("Inserisci la tua età: ")
  val eta: Int = readLine().toInt

  println(s"Your name is $firstName $lastName $eta")
}
```

SovrascritturaFile:

```
import java.io.{File, FileWriter}

object SovrascritturaFile extends App
{
  val file = new File("C:\\Users\\utente\\Desktop\\Ciao_mondo.txt")
  val writer = new FileWriter(file)

  try
  {
    writer.write("Ciao mondo!!!")
    println("Scrittura completata!")
  }
  finally
  {
    writer.close()
  }
}
```

Timer:

```
object Timer
{
  var i = 0
  def oPerSecond(callback: () => Unit){
    while(i < 5)
      {
        callback();
        Thread sleep 1000;
        i = i + 1
      }
  }

  def timeFlies(): Unit = {
    println(s"time flies like an arrow...$i")
  }

  def main(args: Array[String]): Unit = {
    oPerSecond(timeFlies)
  }

}
```

Nome della persona:

Person (class):

```
class Person(name: String)
{
  def nome(): Unit = {
    println("La persona si chiama: " + name)
  }
}
```

Demo (object):

```
object Demo
{
    def main(Args: Array[String]): Unit = {
    val person1 = new Person("Alex");
    val person2 = new Person("John");
    person1.nome();
  }
}
```

AnimaleHabitat:

AnimaleCC (class):

```
package AnimaleHabitat

case class AnimaleCC(id: Int, classe: String, idAmbiente: Int)
```

HabitatCC (class):

```
package AnimaleHabitat

case class HabitatCC(id: Int, tipologia: String)
```

EsempioJoinCaseClass:

```
package AnimaleHabitat

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.log4j.{Level, Logger}

object EsempioJoinCaseClass extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  val conf = new SparkConf().setMaster("local[2]").setAppName("Test
Parallelize")
  val sc = new SparkContext(conf)
  val animaleRDD = sc.parallelize(List((1, "cane", 1), (2, "gatto", 2), (3,
"uccello", 1))).map(x => AnimaleCC(x._1, x._2, x._3)).keyBy(_.idAmbiente)

  val ambienteRDD = sc.parallelize(List((1, "foresta"), (2, "casa"))).map(x =>
HabitatCC(x._1, x._2)).keyBy(_.id)

  val joinAniAmbiRDD = animaleRDD.join(ambienteRDD).map(r => (r._2._1.classe,
r._2._2.tipologia))
  //(1,(AnimaleCC(1,cane,1),HabitatCC(1,foresta))) senza il map mi da tutto
questo oggetto con due elementi, l'intero e la lista
  //quindi(r._2._1.classe) dell oggetto mi prendo la lista ._2 poi della lista
(AnimaleCC(1,cane,1),HabitatCC(1,foresta)
  //mi prendo il primo ._1(AnimaleCC(1,cane,1)) e quindi poi di quest altra
lista mi prendo la classe .classe
  // cioe il secondo elemnto di(1,cane,1) cioe cane

  joinAniAmbiRDD.foreach(println)
}
```

EsempioAzioneTake:

```
package AnimaleHabitat

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.log4j.{Level, Logger}

object EsempioAzioneTake extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  val conf = new SparkConf().setMaster("local[2]").setAppName("Test
Parallelize")
  val sc = new SparkContext(conf)
  val animaleRDD = sc.parallelize(List((1, "cane", 1), (2, "gatto", 2), (3,
"uccello", 1))).map(x => AnimaleCC(x._1, x._2, x._3)).keyBy(_.idAmbiente)

  val ambienteRDD = sc.parallelize(List((1, "foresta"), (2, "casa"))).map(x =>
HabitatCC(x._1, x._2)).keyBy(_.id)

  val joinAniAmbi = animaleRDD.join(ambienteRDD).map(r => (r._2._1.classe,
```

```
r._2._2.tipologia))

  // joinAniAmbiRDD.foreach(println)
  val animaleTake = animaleRDD.take(num = 2)
  println(animaleTake.mkString)
}
```

EsempioAzioneFirst:

```
package AnimaleHabitat

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.log4j.{Level, Logger}

object EsempioAzioneFirst extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  val conf = new SparkConf().setMaster("local[2]").setAppName("Test
Parallelize")
  val sc = new SparkContext(conf)
  val animaleRDD = sc.parallelize(List((1, "cane", 1), (2, "gatto", 2), (3,
"uccello", 1))).map(x => AnimaleCC(x._1, x._2, x._3)).keyBy(_.idAmbiente)

  val ambienteRDD = sc.parallelize(List((1, "foresta"), (2, "casa"))).map(x =>
HabitatCC(x._1, x._2)).keyBy(_.id)

  val joinAniAmbiRRD = animaleRDD.join(ambienteRDD).map(r => (r._2._1.classe,
r._2._2.tipologia))

  val animaleFirst = animaleRDD.first()
  println(animaleFirst)
}
```

Ereditarietà:

Person:

```
package Ereditarietà

class Person(Name: String, PhoneNumber: String, EmailAddress: String)
{
  def showInfo(): Unit = {
    println("\nNome: " + s"$Name" + "\n Numero di telefono: " + s"$PhoneNumber"
+ "\nEmail: " + s"$EmailAddress")
  }
}
```

Professor:

```
package Ereditarietà

class Professor(Name: String, PhoneNumber: String, EmailAddress: String, Salary:
Double) extends Person(Name, PhoneNumber, EmailAddress)
{
```

```
  override def showInfo(): Unit = {
    println("Nome professore: " + s"$Name" + "\nNumero di telefono professore: "
+ s"$PhoneNumber" + "\nIndirizzo email professore: " + s"$EmailAddress")
  }

    def printSalary(): Unit = {
      println(s"Salario $Salary")
    }
}
```

Student:

```
package Ereditarietà

class Student(Name: String, PhoneNumber: String, EmailAddress: String,
StudentNumber: String, AverageMark: String) extends Person(Name, PhoneNumber,
EmailAddress)
{
  override def showInfo(): Unit = {
    println("Nome studente: " + s"$Name" + "\nNumero di telefono studente: " +
s"$PhoneNumber" + "\nIndirizzo email studente: " + s"$EmailAddress" + "\nVoto
medio: " + s"$AverageMark")
  }

  def setAverageMark(newAM: Double): Unit = {
    var AverageMark = newAM
  }


}
```

Main:

```
package Ereditarietà

object Main
{
  def main(Args: Array[String]): Unit = {
    val studente = new Student("Giuseppe", "3827621328",
"gfavuzzi@studenti.apuliadigitalmaker.it", "58828", "8.2")
    studente.showInfo()
    studente.setAverageMark(8.4)

    println(" ")

    val professor = new Professor("Francesco", "2718363729",
"fceci@professori.apuliadigitalmaker.it",  2500.5)
    professor.showInfo()
    professor.printSalary()
  }
}
```

Persona:

Person:

```
package Persona

class Person(firstName: String, lastName: String, age: Int, interests: String,
gender: String)
{
  def nome(): Unit = {
    println("Ciao! Sono" + "\nNome: " + s"$firstName" + "\nCognome: " +
s"$lastName" + "\nEtà: " + s"$age" + "\nInteressi: " + s"$interests" +
"\nSesso:" + s"$gender")
  }
}
```

Main:

```
package Persona

object Main
{
  def main(Args: Array[String]): Unit = {
    val persona = new Person("Giacomo", "de Gennaro", 25, "giocare a calcio",
"maschio");
    persona.nome();
  }
}
```

SparkTest:

```
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.log4j.{Level, Logger}

object sparkTest extends App {

    Logger.getRootLogger.setLevel(Level.WARN)

    val conf = new SparkConf().setMaster("local[1]").setAppName("test")
    val sc = new SparkContext(conf)
    val data = Array(1, 2, 3, 4, 5)
    val rdd = sc.parallelize(data)
    rdd.foreach(println)
}
```

EsempioJoin:

```
package TestJoin

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.log4j.{Level, Logger}

object EsempioJoin extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  val conf = new SparkConf().setMaster("local[*]").setAppName("Test
Parallelize")
  val sc = new SparkContext(conf)
```

```
  val animaleRDD = sc.parallelize(List((1, "cane", 1), (2, "gatto", 2), (3,
"uccello", 1))).keyBy(_._3)
  // con KeyBy 3 indico il valore dei terzi elementi della liste, in questo caso
1, 2 e 1

  val ambienteRDD = sc.parallelize(List((1, "foresta"), (2,
"casa"))).keyBy(_._1)
  // con KeyBy 1 indico il valore dei primi elementi della liste, in questo caso
1 e 2

  val joinAniAmbiRDD = animaleRDD.join(ambienteRDD)

  joinAniAmbiRDD.foreach(println)
}
```

TestMap:

EsempioFlatMap:

```
package TestMap

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.log4j.{Level, Logger}

object EsempioFlatMap extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  val conf = new SparkConf().setMaster("local[*]").setAppName("Test
Parallelize")
  val sc = new SparkContext(conf)
  val sequenza = Seq("ITS Project", "Developer project", "Cloud project")

  val operazioneRDD = sc.parallelize(sequenza)

  operazioneRDD.foreach(println)

  val operazioneRDD2 = operazioneRDD.flatMap(f => f.split(" "))
  operazioneRDD2.foreach(println)
}
```

EsempioMap:

```
package TestMap

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.log4j.{Level, Logger}

object EsempioMap extends App
{
  Logger.getRootLogger.setLevel(Level.WARN)

  val conf = new SparkConf().setMaster("local[*]").setAppName("Test
Parallelize")
  val sc = new SparkContext(conf)
  val sequenza = Seq("ITS Project", "Developer project", "Cloud project")
```

```
  val operazioneRDD = sc.parallelize(sequenza)

  val operazioneRDD2 = operazioneRDD.map(f => (f, 1))
  operazioneRDD2.foreach(println)
}
```

Lezione 6:

La normalizzazione dei dati è una tecnica consolidata per la progettazione di una base di dati.

Prevede di evitare la ripetizione dei dati su più tabelle.

Per rendere più semplici le operazioni di scrittura: inserire, modificare o cancellare dati.

Per contro, un database relazionale in cui i dati sono stati normalizzati tende a disperdere il contenuto informativo su più tabelle.

I vantaggi dei database non relazioni aka NoSql (Not Only Sql):

- **Elevata velocità computazionale;**
- **Supporto per la scalabilità orizzontale;**
- **Un elevato livello di disponibilità del servizio;**
- **Schemaless;**
- **Grossi volumi di dati;**
- **Ciao DBAs;**
- **Compatibili con database relazionali.**

Il JSON (Java-Script-Object-Notation) è un formato adatto all'interscambio di dati client/server.

Un Documento è un insieme di coppie chiave-valore.

I documenti hanno uno schema dinamico, cioè che i documenti nella stessa collection non devono avere lo stesso numero di field e i documenti possono contenere diversi tipi di dati.

Comandi & Esercizi:

Avvio un container tramite un docker-composer:

Mongodb = nome container del mongodb;

Mongosh = il comando per lanciare mongoshell;

Docker exec = eseguo un comando mongo shell all'interno del container mongodb.

```
Microsoft Windows [Versione 10.0.19045.2965]
(c) Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\utente>docker exec -it mongodb mongosh
Current Mongosh Log ID: 64994e761c6f014d12deb871
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.10.0
Using MongoDB:          6.0.6
Using Mongosh:          1.10.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

------
   The server generated these startup warnings when booting
   2023-06-26T08:08:16.512+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2023-06-26T08:08:16.512+00:00: /sys/kernel/mm/transparent_hugepage/enabled is 'always'. We suggest setting it to 'never'
   2023-06-26T08:08:16.512+00:00: vm.max_map_count is too low
------

test> db
test
test> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
test> db.createCollection('books')
{ ok: 1 }
test> show collections
books
test> db.books.drop()
true
test> show collections

test> 1 + 1
2
```

Creazione di un dispositivo inserendolo in una collection:

```
mybd> db.createCollection('books')
{ ok: 1 }
mybd> db.books.insertOne({name: "The Great Gatsby"})
{
  acknowledged: true,
  insertedId: ObjectId("6499597d1095188fa2cce8f2")
}
mybd> db.books.insertOne({name: "Don Quixote"})
{
  acknowledged: true,
  insertedId: ObjectId("6499598a1095188fa2cce8f3")
}
mybd> db.books.insertOne({name: "One Hundred Years of Solitude", year: 1967})
{
  acknowledged: true,
  insertedId: ObjectId("649959961095188fa2cce8f4")
}
mybd>
```

Esercizio 1:

Esercizio 1

Connettiti alla console su localhost. Prova a digitare alcune espressioni JavaScript (mongo shell supporta JavaScript):

- Dimmi quanti secondi ci sono in una settimana
- Dimmi quante settimane ci sono in una vita umana di 80 anni.
- arrotondare il risultato del passaggio precedente con la funzione Math.round ()

exprivia

4

```
test> 7 * 24 * 60 * 60
604800
test> 80 * 365.25 /7
4174.285714285715
test> Math.round(80 * 365.25 /7)
4174
test>
```

1.1 -> 7*24*60*60

1.2 -> 80*365.25/7

1.3 -> Math.round(80*365.25/7)

Esercizio 2:

Esercizio 2

1. Crea un db chiamato mydb. Se visualizzi tutti i db, il db appena creato viene elencato?
2. Crea una collection chiamata collection1 e riprova a elencare tutti i db. Questa volta viene elencato?
3. Elimina il db appena creato e verifca se il db effettivamente è stato eliminato.

```
test> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
test> use mydb
switched to db mydb
mydb> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
mydb> db.createCollection('collection1')
{ ok: 1 }
mydb> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
mydb      8.00 KiB
mydb> dp.dropDatabase()
ReferenceError: dp is not defined
mydb> db.dropDatabase()
{ ok: 1, dropped: 'mydb' }
mydb> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
mydb>
```

2.1

db

show dbs

"NO"

2.2

db.createCollection('collection1')

show dbs

"SI"

2.3

db.dropDatabase()

show dbs

"SI"

Esercizio 3:

Esercizio 3

1. Inserire i seguenti documenti in una collection chiamata pets
2. Usa find per elencare tutti i pets.
3. Trova l'ID di Mikey
4. Trova tutte le creature di nome Mikey che sono Gerbil.
5. Trova tutte le creature con la stringa "dog" nella loro specie.
6. Trova tutte le creature con nome Terrence e la stringa "dog" nella loro specie.
7. Trova tutte le creature con la loro specie inizia con "dog"

name	species
Mikey	Gerbil
Davey	Piranha
Suzy B	Cat
Mikey	Hotdog
Terrence	Sausagedog
Thilomena	Cat
AAAA	Dogcat

```
test> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
test> use mydb
switched to db mydb
mydb> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
mydb> db.createCollection('collection1')
{ ok: 1 }
mydb> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
mydb      8.00 KiB
mydb> dp.dropDatabase()
ReferenceError: dp is not defined
mydb> db.dropDatabase()
{ ok: 1, dropped: 'mydb' }
mydb> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
mydb>
```

```
mydb> db.createCollection('pets')
{ ok: 1 }
mydb> db.pets.insertOne({name: 'Mikey', species: 'Gerbil'})
{
  acknowledged: true,
  insertedId: ObjectId("64995cdb1095188fa2cce8f8")
}
mydb> db.pets.insertOne({name: 'Davey', species: 'Piranha'})
{
  acknowledged: true,
  insertedId: ObjectId("64995d061095188fa2cce8f9")
}
mydb> db.pets.insertOne({name: 'Suzy B', species: 'Cat'})
{
  acknowledged: true,
  insertedId: ObjectId("64995d171095188fa2cce8fa")
}
mydb> db.pets.insertOne({name: 'Mikey', species: 'Hotdog'})
{
  acknowledged: true,
  insertedId: ObjectId("64995d251095188fa2cce8fb")
}
mydb> db.pets.insertOne({name: 'Terrence', species: 'Sausagedog'})
{
  acknowledged: true,
  insertedId: ObjectId("64995d6d1095188fa2cce8fc")
}
mydb> db.pets.insertOne({name: 'Thilomena', species: 'Cat'})
{
  acknowledged: true,
  insertedId: ObjectId("64995d861095188fa2cce8fd")
}
mydb> db.pets.insertOne({name: 'AAAA', species: 'Dogcat'})
{
  acknowledged: true,
  insertedId: ObjectId("64995d9b1095188fa2cce8fe")
}
mydb> db.pets.find()
[
mydb>
    _id: ObjectId("64995cdb1095188fa2cce8f8"),
mydb>
    species: 'Gerbil'
mydb>
  {
mydb>
    name: 'Davey',
mydb>
  },
mydb>
    _id: ObjectId("64995d171095188fa2cce8fa"),
mydb>
    species: 'Cat'
mydb>
  {
mydb>
    name: 'Mikey',
mydb>
  },
mydb>
```

```
mydb>
    species: 'Cat'
mydb>
  {
mydb>
    name: 'Mikey',
mydb>
  },
mydb>
    _id: ObjectId("64995d6d1095188fa2cce8fc"),
mydb>
    species: 'Sausagedog'
mydb>
  {
mydb>
    name: 'Thilomena',
mydb>
  },
mydb>
    _id: ObjectId("64995d9b1095188fa2cce8fe"),
mydb>
    species: 'Dogcat'
mydb>
```

```
mydb> db.pets.find({name: "Mikey"})
[ooks
  {s
    _id: ObjectId("64995cdb1095188fa2cce8f8"),
    name: 'Mikey',
    species: 'Gerbil'
  },
  {
    _id: ObjectId("64995d251095188fa2cce8fb"),
    name: 'Mikey',
    species: 'Hotdog'
  }
]
mydb> show collections
books
pets
mydb> db.pets.find({name: "Mikey", species: "Gerbil"})
[
  {
    _id: ObjectId("64995cdb1095188fa2cce8f8"),
    name: 'Mikey',
    species: 'Gerbil'
  }
]
mydb> db.pets.find({species: /dog/})
[
  {
    _id: ObjectId("64995d251095188fa2cce8fb"),
    name: 'Mikey',
    species: 'Hotdog'
  },
  {
    _id: ObjectId("64995d6d1095188fa2cce8fc"),
    name: 'Terrence',
    species: 'Sausagedog'
  }
]
mydb> db.pets.find({name: "Terrence", species: /dog/})
[
  {
    _id: ObjectId("64995d6d1095188fa2cce8fc"),
    name: 'Terrence',
    species: 'Sausagedog'
  }
]
mydb> db.pets.find({name: "Terrence", species: /^Dog/})

mydb> db.pets.find({species: /^Dog/})
[
  {
    _id: ObjectId("64995d9b1095188fa2cce8fe"),
    name: 'AAAA',
    species: 'Dogcat'
  }
]
mydb>
```

3.1

db.pets.insertOne({name: "Mikey", species: "Gerbil"})

db.pets.insertOne({name: "Davey", species: "Piranha"})

db.pets.insertOne({name: "Suzy B", species: "Cat"})

db.pets.insertOne({name: "Mikey", species: "Hotdog"})

db.pets.insertOne({name: "Terrence", species: "Sausagedog"})

db.pets.insertOne({name: "Philomena", species: "Cat"})

3.2

db.pets.find()

3.3

db.pets.find({name:"Mikey"})

3.4

db.pets.find({name:"Mikey", species: "Gerbil"})

3.5

db.pets.find({species: /dog/})

3.6

db.pets.find({name: "Terrence", species: /dog/})

Esercizio 4:

wget "http://nicholasjohnson.com/mongo/datasets/stocks.json" -o stocks.json

Esercizio 4

Il file json delle azioni è un elenco di tutte le azioni scambiate negli Stati Uniti nel 2015.

http://nicholasjohnson.com/mongo/datasets/stocks.json

1. Scarica e importa il file in un db chiamata stocks e in una collection chiamata stock

Possiamo importare un file JSON dalla riga di comando utilizzando il comando mongoimport.

```
mongoimport --db <nome DB>--collection <nome collection> --file stocks.json
```

2. Visualizza la collection appena importata in una app con GUI (Mongo Compass)
3. Trova tutte le azioni
4. Trova tutte le azioni in cui il (Profit Margin) è superiore a 0,5
5. Trova tutte le azioni con (EPS growth next year) negativa

meet.google.com sta condividendo il tuo schermo. Interrompi condivisione Nascondi


```
Copyright (C) Microsoft Corporation. Tutti i diritti riservati.
                                                                                    Prova la nuova PowerShell multipiattaforma https://aka.ms/pscore6
                                                                                    PS C:\Users\utente> wget "http://nicholasjohnson.com/mongo/datasets/stocks.json" -o stocks.json
Users\utente> mongoimport --db <mydb>-- collection <pets> --file stocks.json
In riga:1 car:18
+ mongoimport --db <mydb>-- collection <pets> --file stocks.json
+                  ~
Operatore '<' riservato per utilizzi futuri.
In riga:1 car:38
+ mongoimport --db <mydb>-- collection <pets> --file stocks.json
+                                      ~
Operatore '<' riservato per utilizzi futuri.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : RedirectionNotSupported

PS C:\Users\utente> docker exec -it mongodb mongoimport --db stocks --collection stock --file /data/db/mongodb-data/stocks.json
2023-06-26T10:53:47.446+0000    Failed: open /data/db/mongodb-data/stocks.json: no such file or directory
2023-06-26T10:53:47.446+0000    0 document(s) imported successfully. 0 document(s) failed to import.
PS C:\Users\utente> docker exec -it mongodb mongoimport --db stocks --collection stock --file /data/db/stocks.json      2023-06-26T10:54:08.368+0000    connected to: mongodb://localhost/
2023-06-26T10:54:09.094+0000    6756 document(s) imported successfully. 0 document(s) failed to import.
PS C:\Users\utente>
```

```
PS C:\Users\utente\Desktop> cd "Scala Docker"
test> show collections
petsent Mongosh Log ID: 6499767cb9228cf9cdaa7b74
test> show collections
pets0
test> use stocks
switched to db stocks   1.10.0
stocks> db.stock.find()
[or mongosh info see: https://docs.mongodb.com/mongodb-shell/
  {
    _id: ObjectId("52853800bb1177ca391c1804"),
    Ticker: 'AAME',
    'Profit Margin': 0.056,
    'Institutional Ownership': 0.05,
    'EPS growth past 5 years': -0.167,
    'Total Debt/Equity': 0.42,
    'Return on Assets': 0.027,
    Sector: 'Financial',
    'P/S': 0.57,
    'Change from Open': -0.0025,
    'Performance (YTD)': 0.316,
    'Performance (Week)': 0,
    'P/B': 0.87,
    'EPS growth quarter over quarter': 7.667,
    'Payout Ratio': 0.05,
    'Performance (Quarter)': 0.0576,
    'P/E': 10.63,
    '200-Day Simple Moving Average': 0.0845,
    'Shares Outstanding': 21.27,
    'Earnings Date': ISODate("2013-11-04T05:00:00.000Z"),
    '52-Week High': -0.0382,
    'P/Cash': 1.82,
    Change: -0.0025,
    'Volatility (Week)': 0.0239,
    Country: 'USA',
    'Return on Equity': 0.082,
```

```
stocks> db.stock.find({"Profit Margin":  {$gt:0.05}  },{"EPS growth this year":true,"EPS growth next year":true})
[
  {
    _id: ObjectId("52853800bb1177ca391c1804"),
    'EPS growth this year': 0.5
  },
  {
    _id: ObjectId("52853800bb1177ca391c17ff"),
    'EPS growth next year': 0.1194,
    'EPS growth this year': 0.147
  },
  {
    _id: ObjectId("52853800bb1177ca391c1807"),
    'EPS growth next year': 0.1649,
    'EPS growth this year': 1
  },
  {
    _id: ObjectId("52853800bb1177ca391c1805"),
    'EPS growth next year': 0.0926,
    'EPS growth this year': 0.573
```

4.1

step1. wget http://nicholasjohnson.com/mongo/datasets/stocks.json -O stocks.json

step2. copiare il file nella cartella ./mongodb-data/json

step3. docker exec -it mongodb mongoimport --db stocks --collection stock --type json --file /data/db/json/stocks.json

4.2

db.stock.find()

4.3

db.stock.find({"Profit Margin" :{ $gt: 0.5}})

4.4

db.stock.find({"EPS growth next year" :{ $lt: 0}})

Esercizio 5:

Esercizio 5

1. Usando projection, mostra soltanto "EPS growth this year" e "EPS growth next year" nella stock collection
2. Usando la soluzione dell' esercizio 4,4, scrivi un comando che filtra il risultato dell' esercizio e mostra soltanto "EPS growth this year" e "EPS growth next year".
3. Usando projection, elimina _id dal risultato.

```
stocks> db.stock.find({"EPS growth this year" : {$gt: 0.5},  "EPS growth next year" : {$gt: 0.5} } , { "EPS growth this year":true, "EPS growth next year" : true, _id:false})
[
  { "EPS growth next year" : 38.52,  "EPS growth this year" : 0.833 },
  { "EPS growth next year" : 2.7538, "EPS growth this year" : 8.25 },
  { "EPS growth next year" : 0.7037, "EPS growth this year" : 2.75 },
  { "EPS growth next year" : 0.658,  "EPS growth this year" : 4.053 },
  { "EPS growth next year" : 0.692,  "EPS growth this year" : 0.773 },
  { "EPS growth next year" : 0.8411, "EPS growth this year" : 2.5 },
  { "EPS growth next year" : 77.9583, "EPS growth this year" : 0.54 },
  { "EPS growth next year" : 14.5,   "EPS growth this year" : 1.02 },
  { "EPS growth next year" : 0.5519, "EPS growth this year" : 0.975 },
  { "EPS growth next year" : 1.3125, "EPS growth this year" : 1.7 },
  { "EPS growth next year" : 1.765,  "EPS growth this year" : 0.786 },
  { "EPS growth next year" : 1.013,  "EPS growth this year" : 0.573 },
  { "EPS growth next year" : 1.0667, "EPS growth this year" : 0.857 },
  { "EPS growth next year" : 0.5257, "EPS growth this year" : 0.781 },
  { "EPS growth next year" : 0.6322, "EPS growth this year" : 2.882 },
  { "EPS growth next year" : 0.506,  "EPS growth this year" : 0.758 },
  { "EPS growth next year" : 112,  "EPS growth this year" : 0.946 },
  { "EPS growth next year" : 0.8182, "EPS growth this year" : 0.761 },
  { "EPS growth next year" : 3.355,  "EPS growth this year" : 0.726 },
  { "EPS growth next year" : 2.077,  "EPS growth this year" : 0.952 }
stocks>
Type "it" for more
```

```
stocks> db.stock.find({"Profit Margin":  {$gt:0.05}  },{"EPS growth this year":true,"EPS growth next year":true,_id:false})
[
  { "EPS growth this year" : 0.5 },
  { "EPS growth next year" : 0.1194, "EPS growth this year" : 0.147 },
  { "EPS growth next year" : 0.1649, "EPS growth this year" : 1 },
  { "EPS growth next year" : 0.0926, "EPS growth this year" : 0.573 },
  { "EPS growth next year" : 0.2562, "EPS growth this year" : 0.022 },
  { "EPS growth next year" : 0.1, "EPS growth this year" : -0.1 },
  { "EPS growth next year" : -0.1053, "EPS growth this year" : 0 },
  { "EPS growth next year" : 0.283, "EPS growth this year" : 0.343 },
  { "EPS growth next year" : 0.0654, "EPS growth this year" : 1.567 },
  { "EPS growth next year" : 0.3992, "EPS growth this year" : 1.121 },
  { "EPS growth next year" : 0.1345, "EPS growth this year" : -0.145 },
  { "EPS growth next year" : 0.0172, "EPS growth this year" : 0.539 },
  { "EPS growth next year" : 0.6192, "EPS growth this year" : -0.395 },
  { "EPS growth next year" : 0.1905, "EPS growth this year" : -0.076 },
  { "EPS growth next year" : 2.7538, "EPS growth this year" : 8.25 },
  { "EPS growth next year" : 0.2, "EPS growth this year" : 1.393 },
  { "EPS growth next year" : 0.1087, "EPS growth this year" : -0.881 },
  { "EPS growth next year" : 0.1333, "EPS growth this year" : 0.217 },
  { "EPS growth next year" : 0.1228, "EPS growth this year" : 0.68 },
  { "EPS growth next year" : 0.3723, "EPS growth this year" : 0.07 }
]
Type "it" for more
stocks>
```

5.1

db.stock.find({},

{"EPS growth this year":true,

"EPS growth next year":true})

.pretty()

5.2

db.stock.find({"EPS growth next year" :{ $lt: 0}},{"EPS growth this year":true,"EPS growth next year":true})

5.3

db.stock.find({"EPS growth next year" :{ $lt: 0}},{"EPS growth this year":true,"EPS growth next year":true, _id:false})

Altri comandi:

docker exec -it mongodb mongoimport --db <stocks>--collection <stocks> --file stocks.json

db.stock.find({"EPS growth this year": {$gt: 0.5}, "EPS growth next year": {$gt: 0.5} } , {"EPS growth this year":true, "EPS growth next year": true, _id:false})

stocks> db.stock.find({},{"EPS growth this year": false, "EPS growth this year": false})

db.stock.find({},{"EPS growth this year":true,"EPS growth next year":true})

db.stock.find({"Profit Margin": {$gt:0.05} },{"EPS growth this year":true,"EPS growth next year":true})

db.stock.find({"Profit Margin": {$gt:0.05} },{"EPS growth this year":true,"EPS growth next year":true,_id:false})

stocks> db.stock.find({"Profit Margin": {$gt:0.05} },{"EPS growth this year":true , "EPS growth next year":true,_id:false})

Lezione 7:

Comandi & Esercizi:

Trovo il libro nella collection dell'anno 1967, dimostrando che esiste:

```
mydb> db.books.find({year: 1967})
[
  {
    _id: ObjectId("64995b9d2d17b40007eb2739"),
    name: 'One Hundred Years of Solitude',
    year: 1967
  }
]
mydb> db.books.find({year: 1968})

mydb> db.books.find({year: { $exists : true} })
[
  {
    _id: ObjectId("64995b9d2d17b40007eb2739"),
    name: 'One Hundred Years of Solitude',
    year: 1967
  }
]
mydb>
```

$lt equivale al minore e $gt equivale al maggiore:

```
mydb> db.books.find({year: {$lt: 1968} })
[
  {
    _id: ObjectId("64995b9d2d17b40007eb2739"),
    name: 'One Hundred Years of Solitude',
    year: 1967
  }
]
```

```
mydb> db.books.find({year: {$lt: 1968}, year: {$gt: 1950} })
[
  {
    _id: ObjectId("64995b9d2d17b40007eb2739"),
    name: 'One Hundred Years of Solitude',
    year: 1967
  }
]
mydb>
```

Altri comandi:

db.stock.find({ "Profit Margin": { $gt: 0.5 } }, { "Profit Margin": true, _id: false })

db.stock.find({ "Profit Margin": { $gt: 0.5 } }, { "Profit Margin": true, _id: false }).sort({"Profit Margin":-1})

db.stock.find({ "Profit Margin": { $gt: 0.5 } }, { "Profit Margin": true, _id: false }).sort({"Profit Margin":-1}).limit(3)

-> Mostra solo i primi 3 numeri

db.stock.find({ "Profit Margin": { $gt: 0.5 } }, { "Profit Margin": true, _id: false }).sort({"Profit Margin":-1}).skip(2).limit(1)

-> Salto i primi 2 e ne mostro uno

db.stock.remove({Country: "France"})

-> Tutti gli stock che hanno France vengono eliminati

db.stock.updateMany(

{Country: "USA"},

{$set:{Country:"United States of America"}}

)

-> Rinominare tutti i campi con il nome "USA" a "united States of America"

db.stock.updateMany(

{Country: "USA"}

{$set:{new_field:1}}

)

-> Creo un nuovo campo

wget "http://nicholasjohnson.com/mongo/datasets/enron.json" -o stocks.json

-> Per accedere al JSON sulla powershell

Esercizio 8:

8.1

wget http://nicholasjohnson.com/mongo/datasets/enron.json -O enron.json

mongoimport --db mydb --collection enron --file enron.json

8.2

db.enron.find({text:/fraud/},{text:1,date:1, _id:false}).sort({"date":1}).pretty()

8.3

db.enron.find({recipients:"mgriffes@evelocity.com"}).count()

8.4

prima bisogna fare una verifica iniziale usando find e poi sostituire find con updateMany e la regola per fare update:

db.enron.find({recipients:/@enron.com/})

db.enron.updateMany({recipients:/@enron.com/},{$set: {recipients:"*@enron.com"}})**

Esercizio 9:

9

db.enron.find({$and:[{text:/fraud/},{text:/yahoo/}]},{text:true, _id:false}).pretty()

Esercizio 10:

Esercizio 10

Nel collection pets,

1. aggiungete un nuovo campo chiamata «legs» con il valore 4
2. aggiungete un nuovo campo chiamata «color» con il valore «white» se «species» è «Cat» altrimenti «Black»

expriVia

37

10.1

db.pets.updateMany(

{},

{"$set": {"legs": 4}})

10.2

db.pets.updateMany(

{species:"Cat"},

{"$set": {"color": "white"}})

prima bisogna fare una verifica e poi usare il filtro nel update:

db.pets.find({ species: {$ne: "Cat"} })

db.pets.updateMany(

{ species: {$ne: "Cat"} },

{"$set": {"color": "Black"}})

Esercizio 11:

Esercizio 11

Usando MongoDB Compass:

1. caricare la seguente tabella con il comando insertOne in db mydb in una collection chiamata continente
2. Trovare tutti i continenti il cui nome inizia con la A
3. Trovare tutti i continenti il cui nome non inizia con la A
4. Trovare tutti i continenti con una popolazione tra 100,000,000 e 20,000,000,000
5. Trovare tutti i continenti il cui nome non inizia con la A o la loro popolazione è tra 100,000,000 e 20,000,000,000

Continente	Popolazione 2020
Asia	4,600,000,000
Africa	1,300,000,000
Europe	700,000,000
North America	600,000,000
South America	500,000,000
Australia/Oceania	50,000,000
Antarctica	null

exprivia

11.1

nel shell di mongo

db.continente.insert({Continente: "Asia", "Popolazione 2020": 4600000000 })

db.continente.insert({continente: "Africa", "Popolazione 2020": 4300000000 })

db.continente.insert({continente: "Europe", "Popolazione 2020": 700000000 })

db.continente.insert({continente: "North America", "Popolazione 2020": 600000000 })

db.continente.insert({continente: "South America", "Popolazione 2020": 500000000 })

db.continente.insert({continente: "Australia/Oceania", "Popolazione 2020": 50000000 })

db.continente.insert({continente: "Antarctica", "Popolazione 2020":null})

in compass

[

{"continente": "Asia", "Popolazione 2020": 4600000000 },
{"continente": "Africa", "Popolazione 2020": 1300000000 },
{"continente": "Europe", "Popolazione 2020": 700000000 },
{"continente": "North America", "Popolazione 2020": 600000000 },
{"continente": "South America", "Popolazione 2020": 500000000 },
{"continente": "Australia/Oceania", "Popolazione 2020": 50000000 },
{"continente": "Antarctica", "Popolazione 2020":null}
]

11.2

```
db.continente.find({continente:/^A/})
```

in compass

```
{continente:/^A/}
```

11.3

```
db.continente.find({continente: {$not: /^A/}})
```

in compass

```
{continente: {$not: /^A/}}
```

11.4

```
db.continente.find({$and: [{"Popolazione 2020":{$lt: 2000000000}},{"Popolazione 2020":{$gt: 100000000}}] })
```

in compass

```
{$and: [{"Popolazione 2020":{$lt: 2000000000}},{"Popolazione 2020":{$gt: 100000000}}] }
```

11.5

```
{$or:
[
{continente: {$not: /^A/}}
,
{$and: [{"Popolazione 2020":{$lt: 2000000000}},{"Popolazione 2020":{$gt: 100000000}}] }
]}
```

Esercizio 12:

Esercizio 12

1. Caricare la seguente collection in una collection chiamata inventory. tags e dim_cm sono array
   - invece di usare 5 volte insertOne potete usare una volta insertmany

   db.collection.insertMany([<document 1> , <document 2>, ...])
2. Trovare tutti gli item con dimensione maggiore di 25
3. Trovare tutti gli item con dimensione tra 20 e 15
4. Trovare tutti gli item con la seconda dimensione (dim_cm.1) maggiore di 20

item	qty	tags	dim_cm
journal	25	blank red	14 21
notebook	50	red blank	14 21
paper	100	red blank plain	14 21
planner	75	blank red	22.85 30
postcard	45	blue	10 15.25

expriVia

46

12.1

```
db.inventory.insertMany( [
  { item: "journal"     , qty:25          ,tags:["blank","red"]           ,dim_cm:[14,21]},
```

```
  { item: "notebook" , qty:50, tags:["red","blank"], dim_cm:[14,21]},
  { item: "paper", qty:100, tags:["red","blank","plain"], dim_cm:[14,21]},
  { item: "planner", qty:75, tags:["blank","red"], dim_cm:[22.85,30]},
  { item: "postcard", qty:45, tags:["blue"], dim_cm:[10,15.25]}
])
```

12.2

```
db.inventory.find({dim_cm:{$gt: 25}})
```

12.3

```
db.inventory.find({dim_cm:{$gt: 15, $lt:20}})
```

12.4

```
db.inventory.find({"dim_cm.1":{$gt: 20}})
```

Esercizio 13:

Esercizio 13

Ripeti l'esercizio 8.4 creando una nuova collection chiamata enron2:
http://nicholasjohnson.com/mongo/datasets/enron.json

1. Sostituisci tutti gli indirizzi email di dominio enron con ***. Ad esempio katherine.brown@enron.com diventa ***@enron.com
2. Metti in confronto il risultato di 8.4 con 13.1

expriria

48

wget "http://nicholasjohnson.com/mongo/datasets/stocks.json" -o enron2.json

-> Punto 1

docker exec -it mongodb mongoimport --db mydb --collection enron2 --file /data/db/enron.json

-> Punto 2

```
db.enron2.updateMany(
   {},
   {$set:{"recipients.$[element]": "***@enron.com" }},
   {arrayFilters:[{element:/@enron.com$/}]}
   )
db.enron2.find({ recipients: /@enron.com/ }, { recipients: true })
```

-> Punto 3

Esercizio 14:

Esercizio 14

1. Caricare la seguente collection in una collection chiamata studenti. Skills è un array
2. Nel campo genere rinomina il valore “m” come “maschio”
3. A chi conosce “c++” o “c#” aggiungi “c” come un nuovo skill
4. Mostra gli studenti con età minore di 45 che non conoscono c in ordine crescente di età. Nel risultato va presentato soltanto il nome, l’età e gli skill
5. Ripeti l’esercizio 14.4 usando GUI di Mongo Compass

nome	eta	genere	skill
Mario	25	m	c++ java
Luigi	50	m	c++
Carlo	40	m	c java python
Marta	20	femmina	python
Alessandra	25	femmina	c#
Clara	18	femmina	

14.1

```
db.studenti.insertMany( [
 {
  "nome": "Mario",
  "eta": 25,
  "genere": "m",
  "skill": [
   "c++",
   "java"
  ]
 },
 {
  "nome": "Luigi",
  "eta": 50,
  "genere": "m",
  "skill": [
```

```
    "c++"
  ]
},
{
  "nome": "Carlo",
  "eta": 40,
  "genere": "m",
  "skill": [
    "c",
    "java",
    "python"
  ]
},
{
  "nome": "Marta",
  "eta": 20,
  "genere": "femmina",
  "skill": [
    "python"
  ]
},
{
  "nome": "Alessandra",
  "eta": 25,
  "genere": "femmina",
  "skill": [
    "c#"
  ]
},
{
```

```
    "nome": "Clara",
    "eta": 18,
    "genere": "femmina"
  }
]
)
```

14.2

```
db.studenti.updateMany(
{genere:"m"},
{$set:{genere:"maschio"}}
)
```

14.3

step1: filtrare chi conosce c++ o c#

```
db.studenti.find({ $or: [{ skill: "c++" }, { skill: "c#" }] })
```

step2: updateMany

```
db.studenti.updateMany({ $or: [{ skill: "c++" }, { skill: "c#" }] },
{$push:{"skill":"c"}})
```

14.4

```
db.studenti.find(
{eta:{$lt: 45},skill:{$ne:"c"}},
{nome:1, eta:1, skill:1,_id:0}
).sort({"eta":1})
```

Lezione 8:

Esercizi con IntelliJ IDEA:

Esercizio 1:

Esercizio 1 – Hello world!

1. Creare un codice che stampi «Hello world!» sullo schermo. Usa la funzione System.out.println

expriria

2

```
package elementari

object HelloWorld
{
  def main(args: Array[String]): Unit =
  {
    System.out.println("Hello world!")
  }
}
```

Esercizio 2:

Esercizio 2 – SparkSession

Il punto di accesso a tutte le funzionalità di Spark è la classe SparkSession.

1. Crea una SparkSession usando la seguente documentazione:
https://spark.apache.org/docs/latest/api/scala/org/apache/spark/sql/SparkSession.html

2. Stampa la versione attuale di spark usando la funzione version di SparkSession

expriVia


```
package elementari

import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.SparkSession

object CreazioneSparkSession
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val sparkSes = SparkSession.builder()
      .master("local[*]")
      .appName("CreateSparkSession")
      .getOrCreate();

    println("Spark Version : " + sparkSes.version)
  }
}
```

Esercizio 3:

Esercizio 3 – RDD to Dataframe

3.1. Crea un dataframe(e visualizza a schermo) a partire dalla seguente tabella e convertilo in un DataFrame seguendo questi passaggi:

nome	secondonome	cognome	id	genere	stipendio
Mario		Foglio	36636	M	1580
Michele	De Rosa		40288	M	1890
Susanna		Guglielmo	42114	F	2080
Lucilla	Anna	Gionata	39192	F	6000
Gianni	Matteo	Ferrara		M	0

- Importare oggetto implicts di SparkSession:
  - import spark.implicits._
  - Maggiori info
- Usare il comando Array per creare le sequenze
- Ogni riga della tabella è una sequenza.
  - Val riga1=("Mario", "", "Foglio", "36636", "M", 1580)
- Per rappresentare una tabella basta creare una variabile che contenga una sequenza di righe:
  - Val data = Array (riga1,riga2, ….)
- Usare il metodo spark.sparkContext.parallelize(data) per convertire rdd in una sequenza:
  - val rdd = spark.sparkContext.parallelize(data)
- Usare il metodo toDF per convertire RDD a dataframe:
  - val columns = Array("nome","secondonome","cognome","id","genere","stipendio")
  - val df = rdd.toDF(columns:_*)

3.2. Usare il metodo show per vedere il dataframe

3.3. Usare il metodo printSchema per vedere lo schema del dataframe


```
package elementari

import org.apache.spark.sql.SparkSession
import org.apache.log4j.{Level, Logger}

object ConvertRDD_ToDataFrame
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    import spark.implicits._

    val data = Array(
      ("Mario", "", "Foglio", "36636", "M", 1580),
      ("Michele", "De Rosa", "", "40288", "M", 1890),
      ("Susanna", "", "Guglielmo", "42114", "F", 2080),
      ("Lucilla", "Anna", "Gionata", "39192", "F", 6000),
      ("Gianni", "Matteo", "Ferrara", "", "M", 0)
    )

    val rdd = spark.sparkContext.parallelize(data)

    val columns = Array("nome", "secondonome", "cognome", "id", "genere",
"stipendio")

    val df = rdd.toDF(columns: _*)
    df.show
    df.printSchema()
  }
}
```

Esercizio 4:

Esercizio 4 – Caricare un file csv e un file json

4.1 Leggi i file country_classification.csv nella cartella resources e convertili in un DataFrame. Verifica il risultato stampando il contenuto e la struttura.

- Usa il metodo spark.read.csv
  - Maggiori info

4.2 Leggi il file serie-a-2022.json nella cartella resources settando l'option multiline per Spark. Verifica il risultato stampando il contenuto e la struttura.

- Usa il metodo spark.read.json
  - Maggiori info
  - Maggiori info

country_classification.cs

country_code	country_label
AD	Andorra
AE	United Arab Emirates
AF	Afghanistan
AG	Antigua and Barbuda
AI	Anguilla
AL	Albania

serie-a-2022.json

AwayTeam	Away TeamScore	DateUtc	Group	HomeTeam	Home TeamScore	Location	Match Number	Round Number
Udinese	2	2022-08-13 16:30:00Z	null	Milan	4	San Siro	6	1
Atalanta	2	2022-08-13 16:30:00Z	null	Sampdoria	0	Luigi Ferraris	9	1
Inter	2	2022-08-13 18:45:00Z	null	Lecce	1	Via del mare	5	1
Torino	2	2022-08-13 18:45:00Z	null	Monza	1	Brianteo	7	1


```
package elementari

import org.apache.spark.sql.SparkSession
import org.apache.log4j.{Level, Logger}

object Caricamento_CSV_JSON
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    val df_csv = spark.read
      .options(Map("inferSchema" -> "true", "delimiter" -> ",", "header" -> 
"true"))
      .csv("C:\\Users\\utente\\Desktop\\country_classification.csv")
    df_csv.printSchema()
    df_csv.show()

    val df_json = spark.read
      .option("multiline", "true")
      .json("C:\\Users\\utente\\Desktop\\serie-a-2022.json")
    df_json.printSchema()
    df_json.show(false)
```

```
  }
}
```

Esercizio 5:

Esercizio 5 – Filtrare un dataframe per colonna

5. Riprendi il file serie-a-2022.json e seleziona solo le colonne evidenziate nella tabella sotto in due modi:

5.1 usando il metodo select

- Maggiori info

5.2 usando il metodo drop

- Maggiori info

AwayTeam	Away TeamScore	DateUtc	Group	HomeTeam	Home TeamScore	Location	Match Number	Round Number
Udinese	2	2022-08-13 16:30:00Z	null	Milan	4	San Siro	6	1
Atalanta	2	2022-08-13 16:30:00Z	null	Sampdoria	0	Luigi Ferraris	9	1
Inter	2	2022-08-13 18:45:00Z	null	Lecce	1	Via del mare	5	1
Juventus	null	2022-08-13 18:45:00Z	null	Udinese	null	Friuli	380	38

exprivia


```
package elementari

import org.apache.spark.sql.SparkSession
import org.apache.log4j.{Level, Logger}

object FiltroDataframeColonna
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    val df_json = spark.read
      .option("multiline", "true")
      .json("C:\\Users\\utente\\Desktop\\serie-a-2022.json")
    df_json.printSchema()
    df_json.show(false)

    val df_json_select = df_json.select("AwayTeam", "AwayTeamScore", "HomeTeam",
"HomeTeamScore", "Location", "MatchNumber", "RoundNumber")
    df_json_select.printSchema()
    df_json_select.show(false)
```

```
    val df_json_drop = df_json.drop("DateUtc", "Group")
    df_json_drop.printSchema()
    df_json_drop.show(false)

  }
}
```

Esercizio 6:

Esercizio 6 – Filtrare un dataframe per riga

6. Riprendi il risultato dell' esercizio precedente e usando il metodo filter seleziona le righe con valore AwayTeamScore non uguale a null in due modi:

6.1 usando la funzione isNull sulla colonna

- usa il GUI di IntelliJ per importare le librerie mancanti mentre scrivi il comando
- Maggiori info
- Maggiori info

6.2 usando una espressione SQL

- Maggiori info

AwayTeam	Away TeamScore	HomeTeam	Home TeamScore	Location	Match Number	Round Number
Udinese	2	Milan	4	San Siro	6	1
Atalanta	2	Sampdoria	0	Luigi Ferraris	9	1
Inter	2	Lecce	1	Via del mare	5	1
Juventus	null	Udinese	null	Friuli	380	38

exprivia


```
package elementari

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.col
import org.apache.log4j.{Level, Logger}

object FiltroDataframeRiga
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    val df_json = spark.read
      .option("multiline", "true")
      .json("C:\\Users\\utente\\Desktop\\serie-a-2022.json")
    df_json.printSchema()
    df_json.show(false)
```

```
    //  val df_json_select =
df_json.select("AwayTeam","AwayTeamScore","HomeTeam","HomeTeamScore","Location",
"MatchNumber","RoundNumber")
    //  df_json_select.printSchema()
    //  df_json_select.show(false)

    val df_json_drop = df_json.drop("DateUtc", "Group")
    //    //  df_json_drop.printSchema()
    //    //  df_json_drop.show(false)
    val df_json_filter1 = df_json_drop.filter(!col("AwayTeamScore").isNull)
    df_json_filter1.show

    val df_json_filter2 = df_json_drop.filter(("AwayTeamScore is not null"))
    df_json_filter2.show
  }
}
```

Esercizio 7:

Esercizio 7 – withColumn e withColumnRenamed

7.1. Al dataframe dell' esercizio precedente aggiungi la colonna GoalDifference con la seguenta formula:

GoalDifference = HomeTeamScore-AwayTeamScore

- Usa il metodo withColumn: Maggiori info

7.2. Rinomina la colonna Location in Stadium

- Usa il metodo withColumnRenamed: Maggiori info


```
package elementari

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.col
import org.apache.log4j.{Level, Logger}

object WithColumn_WithColumnRenamed
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
```

```
      .master("local[*]")
      .getOrCreate();

    val df_json = spark.read
      .option("multiline", "true")
      .json("C:\\Users\\utente\\Desktop\\serie-a-2022.json")
    //  df_json.printSchema()
    //  df_json.show(false)

    val df_json_drop = df_json.drop("DateUtc", "Group")
    val df_json_filter1 = df_json_drop.filter(!col("AwayTeamScore").isNull)
    //  df_json_filter1.show

    val df_withColumn = df_json_filter1.withColumn("GoalDifference",
col("HomeTeamScore") - col("AwayTeamScore"))
    df_withColumn.show

    val df_withColumnRenamed = df_withColumn.withColumnRenamed("Location",
"Stadium")
    df_withColumnRenamed.show
  }
}
```

Esercizio 8:

Esercizio 8 – When

8. All' esercizo precedente aggiungi la colonna Winner che contiene il nome della squadra vincente e null in caso di pareggio.
   - Usa una combinazione di withColumn e When+otherwise
   - Maggiori info sul metodo when+otherwise : Maggiori info


```
package elementari

import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.sql.functions.{col, when}
import org.apache.log4j.{Level, Logger}

object When
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    val df_json = spark.read
      .option("multiline", "true")
      .json("C:\\Users\\utente\\Desktop\\serie-a-2022.json")
    //  df_json.printSchema()
    //  df_json.show(false)

    val df_json_drop = df_json.drop("DateUtc", "Group")
    val df_json_filter1 = df_json_drop.filter(!col("AwayTeamScore").isNull)
    //df_json_filter1.show

    val df_withColumn = df_json_filter1.withColumn("GoalDifference",
col("HomeTeamScore") - col("AwayTeamScore"))
    //df_withColumn.show

    val df_withColumnRenamed = df_withColumn.withColumnRenamed("Location",
"Stadium")
    //df_withColumnRenamed.show

    val df_when = df_withColumnRenamed.withColumn("Winner",
      when(col("GoalDifference") > 0, col("HomeTeam")).
        when(col("GoalDifference") < 0, col("AwayTeam")).
        otherwise(null)
    )
    df_when.show
  }
}
```

Esercizio 9:

Esercizio 9 – Union

9. (non continuare dall' esercizio precedente) Crea un dataframe che contenga i risultati della serie A per il 2021 e il 2022 (serie-a-2021.json e serie-a-2022.json)

- Usa il metodo union per unire i dataframe : Maggiori info
- A ogni dataframe aggiungi la colonna Year= 2021 o 2022
  - Per associare un constant/literal, usa lit: : Maggiori info


```
package elementari

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.{col, lit, when}
import org.apache.log4j.{Level, Logger}

object Union
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    val df_2022 = spark.read
      .option("multiline", "true")
      .json("C:\\Users\\utente\\Desktop\\serie-a-2022.json")
      .withColumn("Year", lit("2022"))
    //df_2022.show

    val df_2021 = spark.read
      .option("multiline", "true")
      .json("C:\\Users\\utente\\Desktop\\serie-a-2021.json")
      .withColumn("Year", lit("2021"))
    //df_2021.show

    val df_serie_a = df_2021.union(df_2022)
    df_serie_a.show
  }
}
```

Esercizio 10:

Esercizio 10 – write

10.1 Scrivi il risultato dell' esercizio precedente in un file csv chiamato serie_a.csv

- Usa il parametro .mode(SaveMode.Overwrite) per poter sovrascrivere il file.

10.2 Rileggi il file csv appena creato e verifica che la struttura e il contenuto siano correttamente creati.

- Usa l'opzione .option("header","true") per poter salvare la testata insieme con il contenuto.


```
package elementari

import org.apache.spark.sql.{SaveMode, SparkSession}
import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.functions.lit

object Write
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
      .master("local")
      .appName("SerieAResults")
      .getOrCreate()

    val df_2021 = spark.read.option("multiline",
"true").json("C:\\Users\\utente\\Desktop\\serie-a-2021.json")
    val df_2022 = spark.read.option("multiline",
"true").json("C:\\Users\\utente\\Desktop\\serie-a-2022.json")

    val df_2021_withYear = df_2021.withColumn("Year", lit(2021))
    val df_2022_withYear = df_2022.withColumn("Year", lit(2022))

    val df_combined = df_2021_withYear.union(df_2022_withYear)

    // Mi scrive il file csv all'interno del folder output all'interno del
progetto scala_advanced
```

```
    df_combined.write.mode(SaveMode.Overwrite).csv("output/serie_a.csv")

    val df_read = spark.read.option("header", "true").csv("output/serie_a.csv")
    df_read.show(false)

    spark.stop()
  }
}
```

Esercizio 11:

Esercizio 11 - groupby

11.1. Carica in un dataframe gli ultimi dati provinciali del COVID da questo percorso:
- ⚠50 MB https://github.com/pcm-dpc/COVID-19/raw/master/dati-json/dpc-covid19-ita-province.json

11.2. Verifica il contenuto del dataframe. Cosa contiene una riga?

11.3. Crea un dataframe che contenga il totale dei casi di ogni regione
- Maggiori info

11.4. Crea un dataframe che contenga il totale dei casi di ogni regione e provincia


```
package elementari

import org.apache.spark.sql.{SaveMode, SparkSession}
import org.apache.spark.sql.functions._
import org.apache.log4j.{Level, Logger}

object GroupBy
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    val df_covid = spark.read
      .option("multiline", "true")
      .json("C:\\Users\\utente\\Desktop\\dpc-covid19-ita-province.json")
```

```
    df_covid.show

    val df_covid_groupby1 = df_covid
      .groupBy("denominazione_regione")
      .agg(sum("totale_casi").as("totale_casi"))

    val df_covid_groupby2 = df_covid
      .groupBy("denominazione_regione", "denominazione_provincia")
      .agg(sum("totale_casi").as("totale_casi"))
    df_covid_groupby2.show

  }
}
```

Lezione 9:

Esercizi con IntelliJ IDEA:

Esercizio 12:

Esercizio 12 - orderBy

12.1 Modifica l'escercizio precedente e ordina i dataframe come segue:

12.1. in ordine alfabetico di regione e provincia

Maggiori info

12.2. in ordine del totale dei casi

expriVia

13

```
package elementari

import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.sql.functions.col
import org.apache.log4j.{Level, Logger}

object OrderBy
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    val df_covid = spark.read
      .option("multiline", "true")
      .json("C:\\Users\\utente\\Desktop\\dpc-covid19-ita-province.json")

    df_covid.show

    val df_covid_groupby1_1 = df_covid.groupBy("denominazione_regione")
      .sum("totale_casi")
      .orderBy(col("denominazione_regione").asc)
    df_covid_groupby1_1.show

    val df_covid_groupby1_2 = df_covid.groupBy("denominazione_regione")
      .sum("totale_casi")
      .orderBy(col("sum(totale_casi)").desc)
    df_covid_groupby1_2.show


    val df_covid_groupby2_1 = df_covid.groupBy("denominazione_regione",
"denominazione_provincia")
      .sum("totale_casi")
      .orderBy(col("denominazione_regione").asc,
col("denominazione_provincia").asc)
    df_covid_groupby2_1.show

    val df_covid_groupby2_2 = df_covid.groupBy("denominazione_regione",
"denominazione_provincia")
      .sum("totale_casi")
      .orderBy(col("sum(totale_casi)").desc)
    df_covid_groupby2_2.show
  }
}
```

Esercizio 13:

Esercizio 13 - join

13. All' esercizio 11.3 aggiungi la popolazione di ogni regione usando il file regioni_italiane_per_popolazione.csv

- Maggiori info


```
package elementari

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.col
import org.apache.log4j.{Level, Logger}

object Join
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    val df_covid = spark.read
      .option("multiline", "true")
      .json("C:\\Users\\utente\\Desktop\\dpc-covid19-ita-province.json")

    //    df_covid.show

    val df_covid_groupby1_1 = df_covid.groupBy("denominazione_regione")
      .sum("totale_casi")
      .orderBy(col("denominazione_regione").asc)
    //    df_covid_groupby1_1.show

    val df_covid_groupby1_2 = df_covid.groupBy("denominazione_regione")
      .sum("totale_casi")
      .orderBy(col("sum(totale_casi)").desc)
    //    df_covid_groupby1_2.show


    val df_covid_groupby2_1 = df_covid.groupBy("denominazione_regione",
"denominazione_provincia")
```

```
      .sum("totale_casi")
      .orderBy(col("denominazione_regione").asc,
col("denominazione_provincia").asc)
    //    df_covid_groupby2_1.show

    val df_covid_groupby2_2 = df_covid.groupBy("denominazione_regione",
"denominazione_provincia")
      .sum("totale_casi")
      .orderBy(col("sum(totale_casi)").desc)
    df_covid_groupby2_2.printSchema()
    df_covid_groupby2_2.show

    val df_pop = spark.read
      .options(Map("inferSchema" -> "true", "delimiter" -> ";", "header" ->
"true"))

.csv("C:\\Users\\utente\\Desktop\\regioni_italiane_per_popolazione_mod.csv")
      .select("Regione", "Popolazione")

    df_pop.printSchema()
    df_pop.show

    val df_join = df_covid_groupby2_2.join(df_pop, df_pop("Regione") ===
df_covid_groupby2_2("denominazione_regione"), "left")
      .withColumnRenamed("Popolazione", "Popolazione_regione")
      .drop("Regione")

    df_join.printSchema()
    df_join.show(999)
  }
}
```

Esercizio 14:

Esercizio 14 - createOrReplaceTempView()

14. Ripeti l'esercizio 12 creando una view temporanea chiamata df_covid_table che contiene la tabella di lettura e ripeti le trasformazioni dell' esercizio 12 usando le query SQL

- Maggiori info


```
package elementari

import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.SparkSession

object CreateOrReplaceTempView
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    val df_covid = spark.read
      .option("multiline", "true")
      .json("C:\\Users\\utente\\Desktop\\dpc-covid19-ita-province.json")

    //    df_covid.show
    //
    //    val df_covid_groupby1_1 = df_covid.groupBy("denominazione_regione")
    //      .sum("totale_casi")
    //      .orderBy(col("denominazione_regione").asc)
    //    df_covid_groupby1_1.show
    //
    //    val df_covid_groupby1_2 = df_covid.groupBy("denominazione_regione")
    //      .sum("totale_casi")
    //      .orderBy(col("sum(totale_casi)").desc)
    //    df_covid_groupby1_2.show
    //
    //
    //    val df_covid_groupby2_1 = df_covid.groupBy("denominazione_regione",
"denominazione_provincia")
    //      .sum("totale_casi")
```

```
    //       .orderBy(col("denominazione_regione").asc,
col("denominazione_provincia").asc)
    //     df_covid_groupby2_1.show
    //
    //     val df_covid_groupby2_2 = df_covid.groupBy("denominazione_regione",
"denominazione_provincia")
    //       .sum("totale_casi")
    //       .orderBy(col("sum(totale_casi)").desc)
    //     df_covid_groupby2_2.show

    df_covid.createOrReplaceTempView("covid")
    spark.sql(
      """
            select
denominazione_regione,denominazione_provincia,sum(totale_casi)
            from covid
            group by denominazione_regione,denominazione_provincia
            order by denominazione_regione,denominazione_provincia
          """).show(999)

    df_covid.createOrReplaceTempView("covid")
    spark.sql(
      """
            select
denominazione_regione,denominazione_provincia,sum(totale_casi)
            from covid
            group by denominazione_regione,denominazione_provincia
            order by sum(totale_casi) desc
          """).show(999)
  }
}
```

Esercizio 15:

Apri docker e avvia il file composer di tipo bat, in modo da accedere a Mongo Express tramite la porta localhost:8081.

Usando il database "mydb", crea una collection chiamata Persone.

All'interno di essa crea un document e inserisci il seguente codice:

[

{

"Nome": "Mario",

"Cognome": "Rossi",

"ID": 1,
"data_di_nascita": {
"anno": 1990,
"mese": 1,
"giorno": 1
}
},
{
"Nome": "Laura",
"Cognome": "Bianchi",
"ID": 2,
"data_di_nascita": {
"anno": 1985,
"mese": 5,
"giorno": 10
}
},
{
"Nome": "Giovanni",
"Cognome": "Verdi",
"ID": 3,
"data_di_nascita": {
"anno": 1995,
"mese": 11,

"giorno": 15

}

}

]

E successivamente inserisci la seguente dependency nel build:

ThisBuild / libraryDependencies += "org.mongodb.spark" %% "mongo-sparkconnector" % "3.0.2"

Infine compila ed esegui il codice di Intellij IDEA.

Esercizio 15 - MongoDB

15.1. Avvia i container di MongoDB e Mongo Express

15.2. Crea una collection chiamata Persone che contenga la seguente Json:

```
[
  {
    "Nome": "Mario",
    "Cognome": "Rossi",
    "ID": 1,
    "data_di_nascita": {
      "anno": 1990,
      "mese": 1,
      "giorno": 1
    }
  },
  {
    "Nome": "Laura",
    "Cognome": "Bianchi",
    "ID": 2,
    "data_di_nascita": {
      "anno": 1985,
      "mese": 5,
      "giorno": 10
    }
  },
  {
    "Nome": "Giovanni",
    "Cognome": "Verdi",
    "ID": 3,
    "data_di_nascita": {
      "anno": 1995,
      "mese": 11,
      "giorno": 15
    }
  }
]
```

Per far funzioare mongo

15.3. Aggiungi il connettore di mongo spark tra i dependecy del sbt e ricarica la sua configurazione.

```
ThisBuild / libraryDependencies += "org.mongodb.spark" %% "mongo-spark-connector" % "3.0.2"
```

15.4. Scrivi un codice che legga la collection «persone»

15.5. Finisci il codice precedente aggiungendo una parte che salva la collection appena letta come persone_copia nella stessa DB di MongoDB

15.6. Verifica il risultato

Maggiori info


```
package elementari

import org.apache.spark.sql.SparkSession
import org.apache.log4j.{Level, Logger}

object MongoDB
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
      .config("spark.master", "local")
      .getOrCreate()

    val df = spark.read.format("mongo")
      .option("spark.mongodb.input.uri", "mongodb://localhost:27018/")
      .option("database", "mydb")
      .option("collection", "Persone")
      .load()

    df.show()

    df.write.format("mongo").mode("overwrite")
      .option("spark.mongodb.output.uri", "mongodb://localhost:27018/")
      .option("database", "mydb")
      .option("collection", "persone_copy")
      .save()

    spark.stop()
  }
}
```

Esercizio 16:

Esercizio 16 - Postgress

16.1. Avvia i container di Postgress e PgAdmin usando la seguenta configurazione

```yaml
version: '3'
services:
  postgres:
    image: postgres
    restart: always
    container_name: postgresql
    environment:
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
      POSTGRES_DB: mydatabase
    ports:
      - '5432:5432'
    volumes:
      - ./postgres-data:/var/lib/postgresql/data
  pgadmin:
    image: dpage/pgadmin4
    restart: always
    container_name: pgadmin
    ports:
      - '8080:80'
    environment:
      PGADMIN_DEFAULT_EMAIL: myemail@example.com
      PGADMIN_DEFAULT_PASSWORD:
mypgadminpassword
      PGADMIN_LISTEN_PORT: 80
    depends_on:
      - postgres
```

expriVia 18

Avviare il compose up di postgress, avviare il container su docker e andare sulla porta localhost:8080.

email: myemail@example.com

password: mypgadminpassword

Successivamente seguire le istruzioni scritte qui:

Esercizio 16 - Postgress

Esercizio 16 - Postgress

16.2. Collegati all' intefaccia grafica di PgAdmin e crea un database chiamato mypg seguendo i passaggi nelle schermate seguente:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tutti i diritti riservati.

Prova la nuova PowerShell multipiattaforma https://aka.ms/pscore6

PS C:\Users\utente> docker ps
CONTAINER ID   IMAGE            COMMAND                  CREATED          STATUS          PORTS                          NAMES
58da1bbd6c28   dpage/pgadmin4   "/entrypoint.sh"         5 minutes ago    Up 5 minutes    443/tcp, 0.0.0.0:8080->80/tcp  pgadmin
62320c97ad8f   postgres         "docker-entrypoint.s…"   5 minutes ago    Up 5 minutes    0.0.0.0:5432->5432/tcp         postgresql
109634bcab71   nginx:latest     "/docker-entrypoint.…"   47 minutes ago   Up 47 minutes   80/tcp                         upbeat_hermann
8c0012152fd5   neo4j:latest     "tini -g -- /startup…"   47 minutes ago   Up 47 minutes   7473-7474/tcp, 7687/tcp        upbeat_wilbur
```

```
PS C:\Users\utente> docker inspect 62320c97ad8f | Select-String -Pattern IPAddress

            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.19.0.2",
```

Nel caso in cui debba rifare la connessione a pgadmin:

```
Microsoft Windows [Versione 10.0.19045.3086]
(c) Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\utente>docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS         PORTS                           NAMES
7a76b8ccb5f3   dpage/pgadmin4 "/entrypoint.sh"         7 minutes ago  Up 7 minutes   443/tcp, 0.0.0.0:8080->80/tcp   pgadmin
f98e024c802e   postgres       "docker-entrypoint.s…"   7 minutes ago  Up 7 minutes   0.0.0.0:5433->5432/tcp          postgresql
109634bcab71   nginx:latest   "/docker-entrypoint.…"   3 hours ago    Up 3 hours     80/tcp                          upbeat_hermann
8c8012152fd5   neo4j:latest   "tini -g -- /startup…"   3 hours ago    Up 3 hours     7473-7474/tcp, 7687/tcp         upbeat_wilbur

C:\Users\utente>docker inspect postgresql
[
    {
        "Id": "f98e024c802efc5933f0b4201684036c8a9ad912c9bf6302ae2b406c22e99447",
        "Created": "2023-06-29T12:45:26.359784Z",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "postgres"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 21394,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2023-06-29T12:45:27.1442641Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:1921ddd0e2c520146bbe1f96f73f1e270baade07da45f761ba4141b2f0c550b7",
        "ResolvConfPath": "/var/lib/docker/containers/f98e024c802efc5933f0b4201684036c8a9ad912c9bf6302ae2b406c22e99447/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/f98e024c802efc5933f0b4201684036c8a9ad912c9bf6302ae2b406c22e99447/hostname",
        "HostsPath": "/var/lib/docker/containers/f98e024c802efc5933f0b4201684036c8a9ad912c9bf6302ae2b406c22e99447/hosts",
        "LogPath": "/var/lib/docker/containers/f98e024c802efc5933f0b4201684036c8a9ad912c9bf6302ae2b406c22e99447/f98e024c802efc5933f0b4201684036c8a9ad912c9bf6302ae2b406c22e99447-json.log",
        "Name": "/postgresql",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": [
                "C:\\Users\\utente\\Desktop\\postgres-data:/var/lib/postgresql/data:rw"
            ],
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
            "NetworkMode": "desktop_default",
            "PortBindings": {
                "5432/tcp": [
                    {
                        "HostIp": "",
                        "HostPort": "5433"
                    }
                ]
            },
```

Esercizio 16 - Postgress

```
PS C:\Users\exprivia> docker ps
CONTAINER ID   IMAGE            COMMAND                  CREATED          STATUS          PORTS                           NAMES
3fc6e110d6f1   dpage/pgadmin4   "/entrypoint.sh"         19 minutes ago   Up 19 minutes   443/tcp, 0.0.0.0:8080->80/tcp   docker_pgadmin_1
91f6a226583e   postgres         "docker-entrypoint.s…"   19 minutes ago   Up 19 minutes   5432/tcp                        docker_postgres_1
PS C:\Users\exprivia> docker inspect 91f6a226583e | Select-String -Pattern IPAddress

            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.19.0.2",
```

Esercizio 16 - Postgress

16.3. Nel database appena costruito, crea una tabella chiamata «persone»

```sql
-- Crea la tabella
CREATE TABLE persone (
  nome VARCHAR(50),
  cognome VARCHAR(50),
  id INT,
  data_di_nascita DATE
);

-- Inserisci i dati nella tabella
INSERT INTO persone (nome, cognome, id, data_di_nascita)
VALUES
  ('Mario', 'Rossi', 1, '1990-01-01'),
  ('Laura', 'Bianchi', 2, '1985-05-10'),
  ('Giovanni', 'Verdi', 3, '1995-11-15');
```


Esercizio 16 - Postgress

16.4. Aggiungi il connettore di Postgress tra i dependecy del sbt e ricarica la sua configurazione.

```
ThisBuild / libraryDependencies += "org.postgresql" % "postgresql" % "42.5.4"
```

16.5. Scrivi un codice che legga la tabella «persone»

16.6. Finisci il codice precedente e aggiungi una parte che salva la tabella appena letta come persone_copia nella stessa DB del Postgress

16.7. Verifica il risultato

Maggiori info

Interrogo il database con queste istruzioni:

-- Crea la tabella

CREATE TABLE persone (

nome VARCHAR(50),

cognome VARCHAR(50),

id INT,

data_di_nascita DATE

);

-- Inserisci i dati nella tabella

INSERT INTO persone (nome, cognome, id, data_di_nascita)

VALUES

('Mario', 'Rossi', 1, '1990-01-01'),

('Laura', 'Bianchi', 2, '1985-05-10'),

('Giovanni', 'Verdi', 3, '1995-11-15');

E controllo se le righe sono state salvate sul dbms di postgress.

Infine compilo ed eseguo il codice di Intellij e dovrebbe uscirci l'output di quello che abbiamo inserito all'interno di postgress:

```
package elementari

import org.apache.spark.sql.SparkSession
import org.apache.log4j.{Level, Logger}

object Postgress
{
  def main(args: Array[String]): Unit =
  {
    Logger.getRootLogger.setLevel(Level.WARN)

    val spark = SparkSession.builder()
      .config("spark.master", "local")
      .getOrCreate()

    val personDF = spark.read
      .format("jdbc")
      .option("url", "jdbc:postgresql://localhost:5433/mydatabase")
      .option("dbtable", "persone")
      .option("user", "myuser")
      .option("password", "mypassword")
      .load()

    val personCopyDF = personDF

    personCopyDF.show()

    personCopyDF.write
```

```
      .format("jdbc")
      .option("url", "jdbc:postgresql://localhost:5433/mydatabase")
      .option("dbtable", "persone_copy")
      .option("user", "myuser")
      .option("password", "mypassword")
      .mode("overwrite")
      .save()

    spark.stop()
  }
}
```

Lezione 10:

Esercizi con IntelliJ IDEA:

Esercizio 1:

Esercizio 1

1.a. Leggendo dalla serie-a-2021.json, crea la tabella della classifica con questo formato:

Squadra	PT
Milan	86
Inter	84
Napoli	79
Juventus	70
Lazio	64
Roma	63
Fiorentina	62

Le colonne in questa tabella rappresentano i seguenti significati:
Squadra: Il nome della squadra di calcio.
PT: Punti totali ottenuti dalla squadra durante la stagione.

Esercizio 1

1.b. Completa la tabella dell' esercizio 1 con le seguenti informazioni

Squadra	PT	PG	V	N	P	GF	GS
Milan	86	38	26	8	4	69	31
Inter	84	38	25	9	4	84	32
Napoli	79	38	24	7	7	74	31
Juventus	70	38	20	10	8	57	37
Lazio	64	38	18	10	10	77	58
Roma	63	38	18	9	11	59	43

Le colonne in questa tabella rappresentano i seguenti significati:

Squadra: Il nome della squadra di calcio.

PT: Punti totali ottenuti dalla squadra durante la stagione.

PG: Partite giocate dalla squadra durante la stagione.

V: Numero di vittorie ottenute dalla squadra durante la stagione.

N: Numero di partite pareggiate dalla squadra durante la stagione.

P: Numero di partite perse dalla squadra durante la stagione.

GF: Goal fatti dalla squadra durante la stagione.

GS: Goal subiti dalla squadra durante la stagione.

a):

```
package it.academy.avanzati

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.{col, when}

object Ava01a_SerieA {

  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    spark.sparkContext.setLogLevel("ERROR")

    val df_json = spark.read
      .option("multiline", "true")
      .json("data/serie-a/serie-a-2021.json")

    val df_json_remove = df_json.drop("DateUtc", "Group", "Location",
"MatchNumber", "RoundNumber")
    //df_json_remove.printSchema()
    //df_json_remove.show(false)

    val df_score = df_json_remove.
      withColumn("HomePt",
        when(col("HomeTeamScore") - col("AwayTeamScore") > 0, 3).
          when(col("HomeTeamScore") - col("AwayTeamScore") < 0, 0).
          otherwise(1)
      ).
```

```
      withColumn("AwayPt",
        when(col("AwayTeamScore") - col("HomeTeamScore") > 0, 3).
          when(col("AwayTeamScore") - col("HomeTeamScore") < 0, 0).
          otherwise(1)
      )

    //  df_score.show

    val df_home = df_score.select(
      col("HomeTeam").as("Squadra"),
      col("HomeTeamScore").as("Goal"),
      col("HomePt").as("PT"))
    val df_away = df_score.select(
      col("AwayTeam").as("Squadra"),
      col("AwayTeamScore").as("Goal"),
      col("AwayPt").as("PT"))

    val df_classifica = df_home.union(df_away)
    df_classifica.show

    val df_finale = df_classifica
      .groupBy("Squadra")
      .sum("PT")
      .withColumnRenamed("sum(PT)", "PT")
      .orderBy(col("PT").desc)

    df_finale.show

  }
}
```

b):

```
package it.academy.avanzati

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._

object Ava01b_SerieA {

  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    spark.sparkContext.setLogLevel("ERROR")

    val df_json = spark.read
      .option("multiline", "true")
      .json("data/serie-a/serie-a-2021.json")

    val df_json_remove = df_json.drop("DateUtc", "Group", "Location",
"MatchNumber", "RoundNumber")
    //df_json_remove.printSchema()
    //df_json_remove.show(false)

    val df_score = df_json_remove.
      withColumn("HomePt",
```

```
        when(col("HomeTeamScore") - col("AwayTeamScore") > 0, 3).
          when(col("HomeTeamScore") - col("AwayTeamScore") < 0, 0).
          otherwise(1)
      ).
      withColumn("AwayPt",
        when(col("AwayTeamScore") - col("HomeTeamScore") > 0, 3).
          when(col("AwayTeamScore") - col("HomeTeamScore") < 0, 0).
          otherwise(1)
      )

    //df_score.show

    val df_home = df_score.select(
      col("HomeTeam").as("Squadra"),
      col("HomeTeamScore").as("GF"),
      col("AwayTeamScore").as("GS"),
      col("HomePt").as("PT"))
    val df_away = df_score.select(
      col("AwayTeam").as("Squadra"),
      col("AwayTeamScore").as("GF"),
      col("HomeTeamScore").as("GS"),
      col("AwayPt").as("PT"))

    val df_classifica = df_home.union(df_away)
      .withColumn("V",
        when(col("PT") === 3, 1)
          .otherwise(0)
      )
      .withColumn("N",
        when(col("PT") === 0, 1)
          .otherwise(0)
      )
      .withColumn("P",
        when(col("PT") === 1, 1)
          .otherwise(0)
      )
    //df_classifica.show

    val df_finale = df_classifica
      .groupBy("Squadra")
      .agg(
        sum("PT").as("PT"),
        count("Squadra").as("PG"),
        sum("V").as("V"),
        sum("P").as("P"),
        sum("N").as("N"),
        sum("GF").as("GF"),
        sum("GS").as("GS")
      )
      .orderBy(col("PT").desc)

    df_finale.show
  }
}
```

Esercizio 2:

Esercizio 2

2.a Per l'esercizio 13 risolvi il problema di mappatura modificando la tabella di lettura

Esercizio 2

2.b Per l'esercizio 13 risolvi il problema di mappatura aggiungendo le righe alla tabella di mappatura

I numeri da usare:

P.A. Bolzano -> 106.951

P.A. Trento -> 117.417

a):

```
package it.academy.avanzati
//Trentino-Alto Adige -> P.A. Bolzano
//Trentino-Alto Adige -> P.A. Trento

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.col

object Ava02a_Geopolitica {

  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    spark.sparkContext.setLogLevel("ERROR")

    val df_covid = spark.read
      .option("multiline", "true")
      .json("data/covid/dpc-covid19-ita-province.json")

    val df_covid_groupby1_1 = df_covid.groupBy("denominazione_regione")
      .sum("totale_casi")
      .orderBy(col("denominazione_regione").asc)
    df_covid_groupby1_1.show

    //  val df_covid_groupby1_2 = df_covid.groupBy("denominazione_regione")
    //    .sum("totale_casi")
    //    .orderBy(col("sum(totale_casi)").desc)
    //  df_covid_groupby1_2.show

    //  val df_covid_groupby2_1 =
df_covid.groupBy("denominazione_regione","denominazione_provincia")
    //    .sum("totale_casi")
    //
.orderBy(col("denominazione_regione").asc,col("denominazione_provincia").asc)
    //  df_covid_groupby2_1.show
    //
    //  val df_covid_groupby2_2 = df_covid.groupBy("denominazione_regione",
"denominazione_provincia")
    //    .sum("totale_casi")
    //    .orderBy(col("sum(totale_casi)").desc)
    //  df_covid_groupby2_2.show

    val df_regioni_pop = spark.read
      .option("header", "true")
      .option("delimiter", ";")
      .csv("data/geopolitica/regioni_italiane_per_popolazione_mod.csv")

    df_regioni_pop.show(100)

    val df_covid_pop = df_covid_groupby1_1.join(df_regioni_pop,
      col("denominazione_regione") === col("Regione"), "left")
      .select(
        col("denominazione_regione"),
        col("sum(totale_casi)"),
        col("Popolazione")
      )
    df_covid_pop.printSchema()
    df_covid_pop.show(100)

  }
}
```

b):

```
package it.academy.avanzati

//Trentino-Alto Adige -> P.A. Bolzano
//Trentino-Alto Adige -> P.A. Trento
//P.A. Bolzano -> 106.951
//P.A. Trento  -> 117.417

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.col

object Ava02b_Geopolitica {

  def main(args: Array[String]): Unit = {
      val spark = SparkSession.builder()
        .master("local[*]")
        .getOrCreate();

    spark.sparkContext.setLogLevel("ERROR")

      import spark.implicits._

      val df_covid = spark.read
        .option("multiline", "true")
        .json("data/covid/dpc-covid19-ita-province.json")

      //  df_covid.show

      val df_covid_groupby1_1 = df_covid.groupBy("denominazione_regione")
        .sum("totale_casi")
        .orderBy(col("denominazione_regione").asc)
      df_covid_groupby1_1.show

      //  val df_covid_groupby1_2 = df_covid.groupBy("denominazione_regione")
      //    .sum("totale_casi")
      //    .orderBy(col("sum(totale_casi)").desc)
      //  df_covid_groupby1_2.show

      //  val df_covid_groupby2_1 =
df_covid.groupBy("denominazione_regione","denominazione_provincia")
      //    .sum("totale_casi")
      //
.orderBy(col("denominazione_regione").asc,col("denominazione_provincia").asc)
      //  df_covid_groupby2_1.show
      //
      //  val df_covid_groupby2_2 = df_covid.groupBy("denominazione_regione",
"denominazione_provincia")
      //    .sum("totale_casi")
      //    .orderBy(col("sum(totale_casi)").desc)
      //  df_covid_groupby2_2.show

      val df_regioni_pop_part1 = spark.read
        .option("header", "true")
        .option("delimiter", ";")
        .csv("data/geopolitica/regioni_italiane_per_popolazione.csv")


      val data = Array(
```

```
        ("P.A. Bolzano", "106951", "", "", "", ""),
        ("P.A. Trento", "117417", "", "", "", "")
      )

      val rdd = spark.sparkContext.parallelize(data)

      val columns = Array("Regione", "Popolazione", "Superficie", "Densità",
"Numero Comuni", "Numero Province")

      val df_regioni_pop_part2 = rdd.toDF(columns: _*)

      val df_regioni_pop = df_regioni_pop_part1.union(df_regioni_pop_part2)

      df_regioni_pop.show(100)


      val df_covid_pop = df_covid_groupby1_1.join(df_regioni_pop,
        col("denominazione_regione") === col("Regione"), "left")
        .select(
          col("denominazione_regione"),
          col("sum(totale_casi)"),
          col("Popolazione")
        )
      df_covid_pop.printSchema()
      df_covid_pop.show(100)
  }
}
```

Esercizio 3:

Esercizio 3

3. Per creare un' applicazione di una libreria online, occorre creare le tabelle di appoggio. Il datamodel da seguire è mostrato nel disegno sotto

3.1. Carica i 5 file csv seguenti come 5 dataframe:
- authors.csv
- books.csv
- borrows.csv
- students.csv
- types.csv

Esercizio 3

3.2. Crea due dataframe che contengano due numeri come segue:

- df_total_books = contiene il numero totale dei libri
- df_total_borrows = contiene il numero totale dei prestiti

Esercizio 3

3.3. Crea due dataframe che contengano il conteggio dei libri in prestito in ordine del conteggio come segue:

df_cnt_bookName

bookName	conteggio
Little Eve Edgarton	88
A Sleep and a Forgetting	53
The Hill of Dreams	51
The Noble Gases	51
Inside The Atom	50
Peace on Earth, Good Will to Dogs	50
A Touch of Sun and Other Stories	49

df_cnt_typeName XL

typeName	conteggio
Diaries	676
Journals	509
Guide	499
Science fiction	447
Biographies	443
Art	433
Romance	369

Esercizio 3

3.4. Crea due dataframe come segue:

- df_nb_students = contiene un elenco degli studenti che non hanno mai preso un libro in prestito.
  - Contiene queste colonne: name, surname
- df_nb_borrows = contiene un elenco dei libri che non sono mai stati presi in prestito
  - Contiene queste colonne: bookName, authorName, authorSurname

Esercizio 3

3.5. Per creare il seguente radar chart, crea un dataframe come segue:

- class_cat è una nuova colonna che contiene la parte numerica della colonna class
- «conteggio» contiene il conteggio dei prestiti
- Deve essere in ordine di typeName e class_cat

df_cnt_typename_class_cat

typeName	class_cat	conteggio
Action and Adventure	9	77
Action and Adventure	10	89
Action and Adventure	11	70
Action and Adventure	12	80
Art	9	113
Art	10	101
Art	11	99
Art	12	120
Biographies	9	105
Biographies	10	112

Esercizio 3

3.6. Crea un dataframe chiamato df_cons_borrows che contiene l'elenco degli studenti che hanno preso più di una volta lo stesso libro in prestito ma tra due prestiti nessun altro ha preso il libro. Il risultato deve contenere nome e cognome dello studente e il titolo del libro.

df_cons_borrows

bookName	name	surname
A Daughter of the Snows	Baker	Bell
Little Dorrit	Annie	Wright
The Four-Fifteen Express	Edwin	Moore
The home-maker	Tatum	Hall
Demons	Albert	Richardson
Sonny: A Christmas Guest	Russell	Wright
Chaturanga	Kelsie	Cox
Hearts of Three	Young	Adams
Gobolinks or Shadow Pictures for Young and Old	Tucker	Cook

Esercizio 3

3.7. Per creare il seguente block chart, crea un dataframe chiamato df_blockchart come segue:

- Il dataframe contiene 3 colonne typeName, authorSurname e bookName
  - typeName: Vanno mostrate solo le prime 9 più frequenti typeName e il resto va mostrato con il nome «Others»
  - authorSurname :A finaco del nome delle colonne tra prantesi va aggiunto il conteggio come segue:
    - typeName (conteggio di tutti i libri per il typeName)
    - authorSurneame (conteggio di tutti i libri in typeName selezionato per authorSurneame)
  - bookName

Hint:
L'operatore «except» in Spark Scala rimuove le righe comuni tra i due DataFrame, restituendo le righe presenti nel DataFrame 1 che non sono presenti nel DataFrame 2.

```
val df3 = df1.except(df2)
```

Esercizio 3

3.7 . Un esempio del dataframe finale:

```
df_blockchart:
+--------------------+--------------+-------------------------------------------+
|typeNameC           |authorSurnameC|bookName                                   |
+--------------------+--------------+-------------------------------------------+
|Journals (14)       |Dickens (2)   |Little Dorrit                              |
|Journals (14)       |Dickens (2)   |A Tale of Two Cities                       |
|Journals (14)       |Brown (1)     |A Plot for Murder                          |
|Journals (14)       |Asimov (2)    |The Near East: 10 000 Years of History     |
|Journals (14)       |Asimov (2)    |Chemistry and Human Health                 |
.
.
.
|Others (59)         |Canfield (1)  |Understood Betsy                           |
|Others (59)         |Dostoevsky (2)|Poor Folk                                  |
|Others (59)         |Dostoevsky (2)|The Idiot                                  |
.
.
.
+--------------------+--------------+-------------------------------------------+
```

Esercizio 3

3.8 . Abbiamo bisogno di creare una tabella di appoggio che rapresenta il bar chart nella immagine. Gli assi si riferiscono al:

- Range del numero di pagine (pagecount)
- Range del punteggio (bookPoint)
  - Se il punteggio è maggiore di 100, va considerato come 100
- Numero dei libri (count bookid)

Esercizio 3

3.9 . Crea una tabella chiamata df_multe che calcola la multa per i ritardi del prestito in questa maniera:

- Il prestito gratuito è consentito per una settimana e se la durata del prestito è maggiore di una settimana bisogna pagare 25 centesimi per ogni giorno di ritardo
- Per la durata del prestito vengono considerati soltanto la data e non l'ora, i minuti e i secondi.
- La tabella deve contenere nome e cognome dello studente e il totale delle multe da pagare per ogni studente.

expriria

a):

```
package it.academy.avanzati

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._
import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.SparkSession

object Ava03a_Libreria {
  def main(args: Array[String]): Unit = {

    Logger.getLogger("org").setLevel(Level.ERROR)
    Logger.getLogger("akka").setLevel(Level.ERROR)

    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    spark.sparkContext.setLogLevel("ERROR")

    // 3.1
    val df_authors = spark.read.option("header",
"true").csv("data/Libreria/authors.csv")
    val df_books = spark.read.option("header",
"true").csv("data/Libreria/books.csv")
    val df_borrows = spark.read.option("header",
"true").csv("data/Libreria/borrows.csv")
    val df_students = spark.read.option("header",
"true").csv("data/Libreria/students.csv")
```

```
    val df_types = spark.read.option("header",
"true").csv("data/Libreria/types.csv")

    println("Authors:");
    df_authors.show(false)
    println("Books:");
    df_books.show(false)
    println("Borrows:");
    df_borrows.show(false);
    df_borrows.printSchema()
    println("Students:");
    df_students.show(false)
    println("Types:");
    df_types.show(false)

    // 3.2
    df_authors.createOrReplaceTempView("authors")
    df_books.createOrReplaceTempView("books")
    df_borrows.createOrReplaceTempView("borrows")
    df_students.createOrReplaceTempView("students")
    df_types.createOrReplaceTempView("types")

    val df_total_books = spark.sql(
      """
      select count(bookId) from books
      """)
//    println("df_total_books:");
//   df_total_books.show(false)

//    val df_total_books2 = df_books.count()
//    println("df_total_books2:" + df_total_books2);

    val df_total_books3 = df_books.groupBy().count()
//    println("df_total_books3:");
//    df_total_books3.show

    val df_total_borrows = spark.sql(
      """
      select count(borrowId) from borrows
      """)
//    println("df_total_borrows:");
//    df_total_borrows.show(false)

    val df_cnt_book_name1 = spark.sql(
      """
          select bookName, count(borrowId) as conteggio
          from borrows
          left join books
          on books.bookId=borrows.bookId
          group by bookName
          order by conteggio desc
          """)
    df_cnt_book_name1.show

    val df_cnt_book_name2 = df_borrows.join(df_books,Seq("bookId"),"left")
      .groupBy("bookName")
      .agg(count("borrowId").as("conteggio"))
      .orderBy(desc("conteggio"))
    df_cnt_book_name2.show

    val df_cnt_typeName1 = spark.sql(
      """
      select typeName, count(borrowId) as conteggio
```

```
    from borrows
    left join books
    on books.bookId=borrows.bookId
    left join types
    on books.typeId=types.typeId
    group by typeName
    order by conteggio desc
    """)
  df_cnt_typeName1.show

  val df_cnt_typeName2 = df_borrows
    .join(df_books, Seq("bookId"), "left")
    .join(df_types, Seq("typeId"), "left")
    .groupBy("typeName")
    .agg(count("borrowId").as("conteggio"))
    .orderBy(desc("conteggio"))
  df_cnt_typeName2.show

  val df_nb_students = df_borrows
    .join(df_students, Seq("studentId"), "full")
    .filter(col("borrowId").isNull)
    .select("name","surname")

  df_nb_students.show

  val df_nb_students2 = df_borrows
    .join(df_students, Seq("studentId"), "full")
    .groupBy("name","surname")
    .agg(count("borrowId").as("conteggio"))
    .filter(col("conteggio")===0)
    .select("name","surname")

  df_nb_students2.show

  val df_nb_students3 = df_borrows
    .join(df_students, Seq("studentId"), "full")
    .groupBy("name", "surname")
    .agg(count("borrowId").as("conteggio"))
    .filter(col("conteggio") === 0)
    .select("name", "surname")

  df_nb_students2.show




  // 3.3
  val df_cnt_bookname = spark.sql(
    """
      select bookName, count(borrowId) as conteggio
      from borrows br
      join books bk
      on br.bookId = bk.bookId
      group by bookName
      order by conteggio desc
      """)
  println("df_cnt_bookname:");
  df_cnt_bookname.show(false)
  val df_cnt_typename = spark.sql(
    """
    select typeName, count(borrowId) as conteggio
    from borrows br
    join books bk
```

```
    on br.bookId = bk.bookId
    join types ty
    on ty.typeId = bk.typeId
    group by typeName
    order by conteggio desc
    """)
  println("df_cnt_typename:");
  df_cnt_typename.show(false)

  // 3.4
  val df_nb_students_ = spark.sql(
    """
    select name,surname, count(borrowId) as conteggio
    from borrows br
    full join students st
    on br.studentId = st.studentId
    group by name,surname
    having conteggio=0
    """)
    .drop("conteggio")
  println("df_nb_students:");
  df_nb_students_.show(false)

  val df_nb_borrows = spark.sql(
    """
  select bookName,authorName,authorSurname, count(borrowId) as conteggio
    from borrows br
    full join books bk
    on br.bookId = bk.bookId
    join authors au
    on au.authorId = bk.authorId
    group by bookName,authorName,authorSurname
    having conteggio=0
  """)
    .drop("conteggio")
  println("df_nb_borrows:");
  df_nb_borrows.show(false)

  // 3.5
  val df_cnt_typename_class_cat = spark.sql(
    """
    select q.typeName, cast(q.class_cat as int), count(borrowId) as conteggio
    from
    (select typeName,
    if(left(class,1)=1,left(class,2),left(class,1)) as class_cat,
    borrowId
    from types ty
    join books bk
    on ty.typeId = bk.typeId
    join borrows br
    on br.bookId = bk.bookId
    join students st
    on br.studentId = st.studentId) q
    group by typeName, class_cat
    order by typeName, class_cat
  """)
  println("df_cnt_typename_class_cat:");
  df_cnt_typename_class_cat.show(false)

  // 3.6
  val df_cons_borrows_step1 = spark.sql(
    """
      select br.bookId, bk.bookName, br.broughtDate, br.studentId, st.name,
```

```
st.surname
       , row_number() OVER (PARTITION BY br.bookId ORDER BY br.bookId,
broughtDate) AS row_num
        from books bk
        join borrows br
        on bk.bookId=br.bookId
        join students st
        on st.studentId=br.studentId
        order by br.bookId,br.broughtDate
      """)

    println("df_cons_borrows_step1:");
    df_cons_borrows_step1.show(false)

    df_cons_borrows_step1.createOrReplaceTempView("cons_borrows_step1")

    val df_cons_borrows = spark.sql(
      """
      select a.bookName, a.name, a.surname
      from cons_borrows_step1 a
      join cons_borrows_step1 b
      on a.bookId=b.bookId and
      a.studentId=b.studentId and
      a.row_num+1 = b.row_num
    """)

    println("df_cons_borrows:");
    df_cons_borrows.show(false)


    // 3.7
    val df_types_group_tmp = spark.sql(
      """
    select bk.typeId, count(bookId) as cnt_types
    from types ty
    join books bk
    on ty.typeId = bk.typeId
    group by bk.typeId
    order by cnt_types desc
  """)
    println("df_types_group_tmp:");
    df_types_group_tmp.show(false)

    val df_types_group_tmp_part1 = df_types_group_tmp.limit(9)
    println("df_types_group_tmp_part1:");
    df_types_group_tmp_part1.show(false)

    val df_types_group_tmp_part2_cnt = df_types_group_tmp
      .except(df_types_group_tmp_part1)
      .groupBy().sum("cnt_types")
      .withColumnRenamed("sum(cnt_types)", "cnt_types")
    //    .withColumn("typeId",lit("Others"))
    println("df_types_group_tmp_part2_cnt:");
    df_types_group_tmp_part2_cnt.show(false)

    val df_types_group_tmp_part2 = df_types_group_tmp
      .except(df_types_group_tmp_part1)
      .select("typeId")
      .join(df_types_group_tmp_part2_cnt)
    println("df_types_group_tmp_part2:");
    df_types_group_tmp_part2.show(false)

    val df_types_group = df_types_group_tmp_part1
```

```
      .withColumn("typeId_flagothers", lit(false))
      .select("typeId", "cnt_types", "typeId_flagothers")
      .union(df_types_group_tmp_part2
        .withColumn("typeId_flagothers", lit(true))
        .select("typeId", "cnt_types", "typeId_flagothers")
      )
    println("df_types_group:");
    df_types_group.show(false)


    val df_authors_group = spark.sql(
      """
      select au.authorId , ty.typeId , count(bookId) as cnt_authors
      from authors au
      join books bk
      on au.authorId = bk.authorId
      join types ty
      on ty.typeId = bk.typeId
      group by au.authorId, ty.typeId
      order by cnt_authors desc, ty.typeId
    """)
    println("df_authors_group:");
    df_authors_group.show(truncate = false, numRows = 999)

    df_types_group.createOrReplaceTempView("types_group")
    df_authors_group.createOrReplaceTempView("authors_group")

    val df_blockchart = spark.sql(
      """
      select
      CONCAT(if(typeId_flagothers=false,typeName,"Others")," (",cnt_types,")")
as typeNameC,
      CONCAT(authorSurname," (",cnt_authors,")") as authorSurnameC,
      bookName
      from types_group tyg
      join types ty
      on ty.typeId = tyg.typeId
      join books bk
      on bk.typeId = tyg.typeId
      join authors au
      on au.authorId = bk.authorId
      join authors_group aug
      on aug.authorId = au.authorId and aug.typeId=bk.typeId
    """)

    println("df_blockchart:");
    df_blockchart.show(numRows = 999, truncate = false)

    //3.8
    val df_book_3d = spark.sql(
      """
      SELECT
        CASE
          WHEN pagecount <= 100 THEN '0-100'
          WHEN pagecount <= 200 THEN '101-201'
          WHEN pagecount <= 300 THEN '201-301'
          WHEN pagecount <= 400 THEN '301-401'
          ELSE '400>'
        END AS pagecount_group,
        CASE
          WHEN bookPoint <= 25 THEN '0-25'
          WHEN bookPoint <= 50 THEN '25-50'
          WHEN bookPoint <= 75 THEN '50-75'
```

```
          ELSE '75-100'
        END AS bookpoint_group,
        COUNT(bookId) AS bookId_count
      FROM
        books
      GROUP BY
        pagecount_group,
        bookpoint_group
    """)
    println("df_book_3d:");
    df_book_3d.show()

    //3.9
    df_borrows.createOrReplaceTempView("borrows")

    val df_borrows_mod = spark.sql("""
    SELECT
        borrowId,
        studentId,
        bookId,
        to_date(takenDate, 'yyyy-MM-dd HH:mm:SS') AS takenDateOK,
        to_date(broughtDate, 'yyyy-MM-dd HH:mm:SS') AS broughtDateOK,
        datediff(to_date(broughtDate, 'yyyy-MM-dd HH:mm:SS'), to_date(takenDate,
'yyyy-MM-dd HH:mm:SS')) AS datesDiff,
        CASE WHEN datediff(to_date(broughtDate, 'yyyy-MM-dd HH:mm:SS'),
to_date(takenDate, 'yyyy-MM-dd HH:mm:SS')) <= 7 THEN 0
            ELSE datediff(to_date(broughtDate, 'yyyy-MM-dd HH:mm:SS'),
to_date(takenDate, 'yyyy-MM-dd HH:mm:SS')) * 0.25
        END AS multa
    FROM
        borrows
""")

    df_borrows_mod.printSchema()
    println("df_borrows_mod:");
    df_borrows_mod.show();
    df_borrows_mod.printSchema()
    df_borrows_mod.createOrReplaceTempView("borrows_mod")

    val df_multe = spark.sql(
      """
        select studentId,name,surname, sum(multa)
        from
        (select st.*,brm.multa
        from
        borrows_mod brm
        join students st
        on brm.studentId = st.studentId) q
        group by studentId,name,surname
      """)

    df_multe.show

  }
}
```

b):

```
package it.academy.avanzati

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.expressions.Window
import org.apache.spark.sql.functions._

object Ava03b_Libreria {
  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder()
      .master("local[*]")
      .getOrCreate();

    spark.sparkContext.setLogLevel("ERROR")

    import spark.implicits._

    // 3.a
    val df_authors = spark.read.option("header",
"true").csv("data/Libreria/authors.csv")
    val df_books = spark.read.option("header",
"true").csv("data/Libreria/books.csv")
    val df_borrows = spark.read.option("header",
"true").csv("data/Libreria/borrows.csv")
    val df_students = spark.read.option("header",
"true").csv("data/Libreria/students.csv")
    val df_types = spark.read.option("header",
"true").csv("data/Libreria/types.csv")

    println("Authors:");
    df_authors.show(false)
    println("Books:");
    df_books.show(false)
    println("Borrows:");
    df_borrows.show(false);
    df_borrows.printSchema()
    println("Students:");
    df_students.show(false)
    println("Types:");
    df_types.show(false)

    // 3.b
    val df_total_books = Seq(df_books.count()).toDF()
    println("df_total_books:")
    df_total_books.show(false)
    val df_total_borrows = Seq(df_borrows.count()).toDF()
    println("df_total_borrows:")
    df_total_borrows.show(false)

    // 3.c
    val df_cnt_bookname = df_borrows
      .join(df_books, df_borrows("bookId") === df_books("bookId"))
      .groupBy("bookName")
      .agg(count("borrowId").alias("conteggio"))
      .orderBy(desc("conteggio"))
    println("df_cnt_bookname:")
    df_cnt_bookname.show(false)

    val df_cnt_typename = df_borrows
      .join(df_books, df_borrows("bookId") === df_books("bookId"))
      .join(df_types, df_books("typeId") === df_types("typeId"))
      .groupBy("typeName")
      .agg(count("borrowId").alias("conteggio"))
```

```
      .orderBy(desc("conteggio"))
    println("df_cnt_typename:")
    df_cnt_typename.show(false)

    // 3.d
    val df_nb_students = df_borrows
      .join(df_students, df_borrows("studentId") === df_students("studentId"),
"full")
      .groupBy("name", "surname")
      .agg(count("borrowId").alias("conteggio"))
      .filter("conteggio = 0")
      .drop("conteggio")
    println("df_nb_students:")
    df_nb_students.show(false)

    val df_nb_borrows = df_borrows
      .join(df_books, df_borrows("bookId") === df_books("bookId"),"full")
      .join(df_authors, df_books("authorId") === df_authors("authorId"),"inner")
      .groupBy("bookName", "authorName", "authorSurname")
      .agg(count("borrowId").alias("conteggio"))
      .filter("conteggio = 0")

    println("df_nb_borrows:")
    df_nb_borrows.show

    // 3.e
    val df_cnt_typename_class_cat = df_types
      .join(df_books, df_types("typeId") === df_books("typeId"))
      .join(df_borrows, df_books("bookId") === df_borrows("bookId"))
      .join(df_students, df_borrows("studentId") === df_students("studentId"))
      .selectExpr("typeName", "IF(LEFT(class, 1) = 1, LEFT(class, 2),
LEFT(class, 1)) AS class_cat", "borrowId")
      .groupBy("typeName", "class_cat")
      .agg(count("borrowId").alias("conteggio"))
      .orderBy("typeName", "class_cat")

    println("df_cnt_typename_class_cat:")
    df_cnt_typename_class_cat.show(false)

    // 3.f
    val windowSpec =
Window.partitionBy(df_books("bookId")).orderBy(df_borrows("broughtDate"))

    val df_cons_borrows_step1 = df_books
      .join(df_borrows, df_books("bookId") === df_borrows("bookId"))
      .join(df_students, df_students("studentId") === df_borrows("studentId"))
      .select(df_books("bookId"), df_books("bookName"),
df_borrows("broughtDate"),
        df_students("studentId"), df_students("name"), df_students("surname"),
        row_number().over(windowSpec).alias("row_num"))
      .orderBy(df_books("bookId"), df_borrows("broughtDate"))

    println("df_cons_borrows_step1:")
    df_cons_borrows_step1.show(false)

    val df_cons_borrows = df_cons_borrows_step1.as("a")
      .join(df_cons_borrows_step1.as("b"), $"a.bookId" === $"b.bookId" &&
$"a.studentId" === $"b.studentId" && $"a.row_num" + 1 === $"b.row_num")
      .select($"a.bookName", $"a.name", $"a.surname")

    println("df_cons_borrows:")
    df_cons_borrows.show(false)
```

```
    // 3.g
    val df_types_group_tmp = df_types.join(df_books, "typeId")
      .groupBy("typeId")
      .agg(count("bookId").alias("cnt_types"))
      .orderBy(desc("cnt_types"))

    println("df_types_group_tmp:")
    df_types_group_tmp.show(false)

    val df_types_group_tmp_part1 = df_types_group_tmp.limit(9)
    println("df_types_group_tmp_part1:")
    df_types_group_tmp_part1.show(false)

    val df_types_group_tmp_part2_cnt = df_types_group_tmp
      .except(df_types_group_tmp_part1)
      .groupBy()
      .agg(sum("cnt_types").alias("cnt_types"))
    //  .withColumn("typeId", lit("Others"))

    println("df_types_group_tmp_part2_cnt:")
    df_types_group_tmp_part2_cnt.show(false)

    val df_types_group_tmp_part2 = df_types_group_tmp
      .except(df_types_group_tmp_part1)
      .select("typeId")
      .join(df_types_group_tmp_part2_cnt)

    println("df_types_group_tmp_part2:")
    df_types_group_tmp_part2.show(false)

    val df_types_group = df_types_group_tmp_part1
      .withColumn("typeId_flagothers", lit(false))
      .select("typeId", "cnt_types", "typeId_flagothers")
      .union(df_types_group_tmp_part2
        .withColumn("typeId_flagothers", lit(true))
        .select("typeId", "cnt_types", "typeId_flagothers")
      )

    println("df_types_group:")
    df_types_group.show(false)

    val df_authors_group = df_authors
      .join(df_books, df_authors("authorId") === df_books("authorId"))
      .join(df_types, df_types("typeId") === df_books("typeId"))
      .groupBy(df_authors("authorId"), df_types("typeId"))
      .agg(count("bookId").alias("cnt_authors"))
      .orderBy(desc("cnt_authors"), df_types("typeId"))

    println("df_authors_group:")
    df_authors_group.show(truncate = false, numRows = 999)

    val df_blockchart = df_types_group
      .join(df_types, Seq("typeId"))
      .join(df_books, Seq("typeId"))
      .join(df_authors, Seq("authorId"))
      .join(df_authors_group, Seq("authorId", "typeId"))
      .withColumn("typeNameC",
        concat(
          when(!col("typeId_flagothers"),
col("typeName")).otherwise(lit("Others")),
          lit(" ("),
          col("cnt_types"),
          lit(")")
```

```
        )
      ).withColumn("authorSurnameC",
      concat(
        col("authorSurname"),
        lit(" ("),
        col("cnt_authors"),
        lit(")")
      )
    )
      .select("typeNameC","authorSurnameC","bookName"  )

    println("df_blockchart:")
    df_blockchart.show(numRows = 999, truncate = false)

    // 3.g
    val df_book_3d = df_books
      .select(
        when(col("pagecount") <= 100, "0-100")
          .when(col("pagecount") <= 200, "101-201")
          .when(col("pagecount") <= 300, "201-301")
          .when(col("pagecount") <= 400, "301-401")
          .otherwise("400>") as "pagecount_group",
        when(col("bookPoint") <= 25, "0-25")
          .when(col("bookPoint") <= 50, "25-50")
          .when(col("bookPoint") <= 75, "50-75")
          .otherwise("75-100") as "bookpoint_group",
        col("bookId") as "bookId"
      )
      .groupBy("pagecount_group", "bookpoint_group")
      .agg(count("bookId") as "bookId_count")

    println("df_book_3d:")
    df_book_3d.show()
 // ta inja ok
 //3.i
 val df_borrows_mod = df_borrows
   .withColumn("takenDateOK", to_date(col("takenDate"), "yyyy-MM-dd HH:mm:SS"))
   .withColumn("broughtDateOK", to_date(col("broughtDate"), "yyyy-MM-dd
HH:mm:SS"))
   .drop("takenDate")
   .drop("broughtDate")
   .withColumnRenamed("takenDateOK", "takenDate")
   .withColumnRenamed("broughtDateOK", "broughtDate")
   .withColumn("datesDiff", datediff(col("broughtDate"), col("takenDate")))
   .withColumn("multa",
     when(col("datesDiff") <= 7, 0)
       .otherwise(col("datesDiff") * 0.25))

    df_borrows_mod.printSchema()
    println("df_borrows_mod:");
    df_borrows_mod.show();
    df_borrows_mod.printSchema()
    df_borrows_mod.createOrReplaceTempView("borrows_mod")


    val df_multe = df_borrows_mod
      .join(df_students, Seq("studentId"))
      .groupBy("studentId", "name", "surname")
      .agg(sum("multa").as("sum_multa"))

    df_multe.show
  }
}
```

Esercizio 4:

Esercizio 4

4.1 Carica le seguenti tabelle in mongoDB:

Nome Database: geopolitica

Nome Collection: worldcities

Download link:
https://gist.githubusercontent.com/curran/13d30e855d48cdd6f22acdf0afe27286/raw/0635f14817ec634833bb904a47594cc2f5f9dbf8/worldcities_clean.csv

Dimensione: 183KB

Nome Database: airbnb

Nome Collection: airbnb

Download link(da caricare il formato json):
https://public.opendatasoft.com/explore/dataset/airbnb-listings/export/?disjunctive.host_verifications&disjunctive.amenities&disjunctive.features

Dimensione: 2,80 GB

⚠invece di usare il file sopra, inzialmente useremo un estratto random del file con 1000 oggetti: airbnb-listings_shuf_1000.json

Esercizio 4

4.2 verifica il contenuto di ogni file importato:

4.2.1 per worldcities conta il numero di righe nei file csv e mettilo in confronto con il numero dei documenti nella collection importata.

4.2.2 Per airbnb-listings.json conta il numero degli oggetti e mettilo in confronto con il numero dei documenti nella collection importata

Esercizio 4

4.3 Crea un'applicazione di Spark scala che legga le due collection indicate e stampa la struttura e un'estrazione di ognuna

Esercizio 4

4.4.1 aggiungi i seguenti campi alla collection di airbnb:

A. "weekly_price": price * 7

B. "average_review_scores": Calcola la media dei punteggi delle recensioni ("review_scores_accuracy", "review_scores_cleanliness", "review_scores_checkin", "review_scores_communication", "review_scores_location" e "review_scores_value") per ottenere un punteggio medio complessivo.

4.4.2 Una volta fatto, verifica il risultato assicurandoti che i nuovi campi siano sempre valorizzati.

4.4.3 Se nel 4.4.2 ci sono campi non valorizzati, scrivi un codice che li mostra per indagare meglio il problema usando i campi relativi nell'output.

Esercizio 4

4.5.1 Carica la collection worldcities. Considerando che contiene le geolocalizzazione (long e lat) del centro città, usa il campo "city" di airbnb come chiave del join per aggiungere tali informazioni e rinominali come city_center_longitude e city_center_latitude:

ecco un esempio:
"id": "6017649",
…
"latitude": "52.358295084395394",
"longitude": "4.859091947726598",
"city_center_latitude : "4.862859671199358",
"city_center_longitude ": 52.35992696894107
….

4.5.2 Una volta fatto, verifica il risultato assicurandoti che i nuovi campi siano sempre valorizzati.

Esercizio 4

4.5.3 Se nel 4.5.2 ci sono campi non valorizzati, scrivi un codice che li mostri per indagare meglio il problema.

4.5.4 ripeti 4.5.3 ma questa volta mostra una lista distinta delle città che non sono mappate correttamente.

Esercizio 4

4.6.1 Usando long a lat aggiunti nello step precedente, crea un nuovo campo chiamato distance_city_center che calcola la distanza di ogni bed&breakfast dal centro città usando questa formula:

$$\sqrt{(city_center_latitude - latitude)^2 + (city_center_longitude - longitude)^2} \times 111.32$$

4.6.2 Controlla il risultato

Info sulla formula 👇

$$\text{distanza euclidea} = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

$$\text{Fattore di conversione} = \frac{\text{Circonferenza media della Terra}}{360} = \frac{40075}{360} \approx 111.32$$

Esercizio 4

4.7 Scrivi il dataframe finale in postgres in un database chiamato airbnb e una collection chiamata airbnb. Occorre convertire come una struttura flat tutti le colonne con struttura di struct.


```
package it.academy.avanzati

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._
import org.apache.log4j.{Level, Logger}

object Ava04_Airbnb {
  def main(args: Array[String]): Unit = {

    Logger.getLogger("org").setLevel(Level.ERROR)
    Logger.getLogger("akka").setLevel(Level.ERROR)

    val spark = SparkSession.builder()
      .config("spark.mongodb.input.uri", "mongodb://localhost:27018/")
      .config("spark.mongodb.output.uri", "mongodb://localhost:27018/")
      .config("spark.master", "local")
      .getOrCreate()

    spark.sparkContext.setLogLevel("ERROR")

    // 4.3
    println("4.3")
    val df_airbnb = spark.read.format("mongo")
      .option("database", "airbnb")
      .option("collection", "airbnb_1000")
      .load()

    df_airbnb.printSchema()
    df_airbnb.show()

    val df_worldcities = spark.read.format("mongo")
      .option("database", "geopolitica")
      .option("collection", "worldcities")
      .load()
```

```
    df_worldcities.printSchema()
    df_worldcities.show()

    // 4.4
    println("4.4.A")
    val df_airbnb_2 = df_airbnb.withColumn("weekly_price",col("price")*7)
        df_airbnb_2.printSchema()
        df_airbnb_2.show()

    println("4.4.B")
    val df_airbnb_3 = df_airbnb_2.withColumn("average_review_scores",
      (col("review_scores_accuracy") +
      col("review_scores_cleanliness") +
      col("review_scores_checkin") +
      col("review_scores_communication") +
      col("review_scores_location") +
      col("review_scores_value")) / 6
    )

    df_airbnb_3.printSchema()
    df_airbnb_3.show()

    println("4.4.2")

    val weekly_price_null =
df_airbnb_3.filter(col("weekly_price").isNull).count()
    println("weekly_price null count: " + weekly_price_null)
    val average_review_scores_null =
df_airbnb_3.filter(col("average_review_scores").isNull).count()
    println("average_review_scores null count: " + average_review_scores_null)

    println("4.4.3")
    println("weekly_price null:")
    df_airbnb_3.filter(col("weekly_price").isNull)
      .select("_id","price","weekly_price")
      .show()
    println("average_review_scores null:")
    df_airbnb_3.filter(col("average_review_scores").isNull)
      .select(col("_id"),
        col("review_scores_accuracy"),
        col("review_scores_cleanliness"),
        col("review_scores_checkin"),
        col("review_scores_communication"),
        col("review_scores_location"),
        col("review_scores_value"))
      .show()

    println("4.5.1")
    val df_worldcities2=df_worldcities.select("city", "lat" , "lng")
      .withColumnRenamed("lat","city_center_latitude")
      .withColumnRenamed("lng","city_center_longitude")
    val df_airbnb_4 = df_airbnb_3.join(df_worldcities2,Seq("city"), "left")
    df_airbnb_4.printSchema()
//    df_airbnb_4.show()
    df_airbnb_4
      .select("_id","city","latitude", "longitude","city_center_latitude" ,
"city_center_longitude")
      .show()
    println("4.5.2")
    val city_center_null = df_airbnb_4
      .filter(col("city_center_latitude").isNull ||
col("city_center_longitude").isNull)
      .count()
```

```
    println("city_center null count: " + city_center_null)

    println("4.5.3")
    df_airbnb_4
      .select("_id","city","latitude", "longitude","city_center_latitude" ,
"city_center_longitude")
      .filter(col("city_center_latitude").isNull ||
col("city_center_longitude").isNull)
      .show

    println("4.5.4")
    df_airbnb_4
      .select("city")
      .filter(col("city_center_latitude").isNull ||
col("city_center_longitude").isNull)
      .dropDuplicates()
      .orderBy("city")
      .show(999)

    println("4.6.1")
    val df_airbnb_5 = df_airbnb_4.withColumn("distance_city_center",
      (sqrt(pow(df_airbnb_4("city_center_latitude") - df_airbnb_4("latitude"),
2) +
        pow(df_airbnb_4("city_center_longitude") - df_airbnb_4("longitude"),
2))) * 111.32)

        df_airbnb_5.printSchema()
        df_airbnb_5

.select("distance_city_center","city_center_latitude","latitude","city_center_lo
ngitude","longitude")
          .show

     4.7
    df_airbnb_5.printSchema()

    import org.apache.spark.sql.functions._

    // Flatten all structs in the DataFrame
    val df_airbnb_6 = df_airbnb_5.select(
      col("city"),
      col("_id.oid").as("_id"),
      col("access"),
      col("accommodates"),
      col("amenities"),
      col("availability_30"),
      col("availability_365"),
      col("availability_60"),
      col("availability_90"),
      col("bathrooms"),
      col("bed_type"),
      col("bedrooms"),
      col("beds"),
      col("calculated_host_listings_count"),
      col("calendar_last_scraped"),
      col("calendar_updated"),
      col("cancellation_policy"),
      col("cleaning_fee"),
      col("country"),
      col("country_code"),
      col("description"),
      col("experiences_offered"),
      col("extra_people"),
```

```
        col("features"),
        col("first_review"),
        col("geolocation.lon").as("geolocation_lon"),
        col("geolocation.lat").as("geolocation_lat"),
        col("guests_included"),
        col("has_availability"),
        col("host_about"),
        col("host_acceptance_rate"),
        col("host_id"),
        col("host_listings_count"),
        col("host_location"),
        col("host_name"),
        col("host_neighbourhood"),
        col("host_picture_url"),
        col("host_response_rate"),
        col("host_response_time"),
        col("host_since"),
        col("host_thumbnail_url"),
        col("host_total_listings_count"),
        col("host_url"),
        col("host_verifications"),
        col("house_rules"),
        col("id"),
        col("interaction"),
        col("jurisdiction_names"),
        col("last_review"),
        col("last_scraped"),
        col("latitude"),
        col("license"),
        col("listing_url"),
        col("longitude"),
        col("market"),
        col("maximum_nights"),
        col("medium_url"),
        col("minimum_nights"),
        col("monthly_price"),
        col("name"),
        col("neighborhood_overview"),
        col("neighbourhood"),
        col("neighbourhood_cleansed"),
        col("neighbourhood_group_cleansed"),
        col("notes"),
        col("number_of_reviews"),
        col("picture_url.thumbnail").as("picture_url_thumbnail"),
        col("picture_url.filename").as("picture_url_filename"),
        col("picture_url.format").as("picture_url_format"),
        col("picture_url.width").as("picture_url_width"),
        col("picture_url.mimetype").as("picture_url_mimetype"),
        col("picture_url.etag").as("picture_url_etag"),
        col("picture_url.id").as("picture_url_id"),
        col("picture_url.last_synchronized").as("picture_url_last_synchronized"),
        col("picture_url.color_summary").as("picture_url_color_summary"),
        col("picture_url.height").as("picture_url_height"),
        col("price"),
        col("property_type"),
        col("review_scores_accuracy"),
        col("review_scores_checkin"),
        col("review_scores_cleanliness"),
        col("review_scores_communication"),
        col("review_scores_location"),
        col("review_scores_rating"),
        col("review_scores_value"),
        col("reviews_per_month"),
```

```
      col("room_type"),
      col("scrape_id"),
      col("security_deposit"),
      col("smart_location"),
      col("space"),
      col("square_feet"),
      col("state"),
      col("street"),
      col("summary"),
      col("thumbnail_url"),
      col("transit"),
      col("weekly_price"),
      col("xl_picture_url"),
      col("zipcode"),
      col("average_review_scores"),
      col("city_center_latitude"),
      col("city_center_longitude"),
      col("distance_city_center")
    )


    df_airbnb_6.write
      .format("jdbc")
      .option("url", "jdbc:postgresql://localhost:5432/mydatabase")
      .option("dbtable", "airbnb")
      .option("user", "myuser")
      .option("password", "mypassword")
      .mode("overwrite")
      .save()
  }
}
```