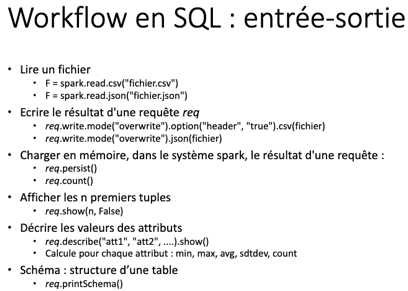
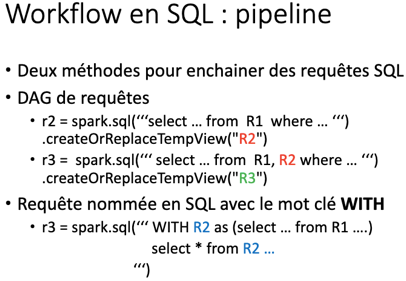
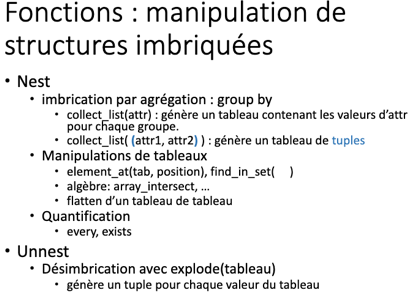
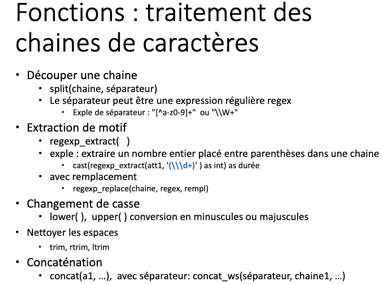
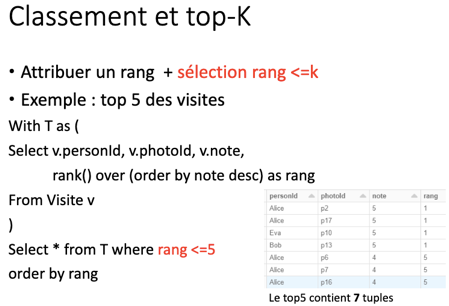
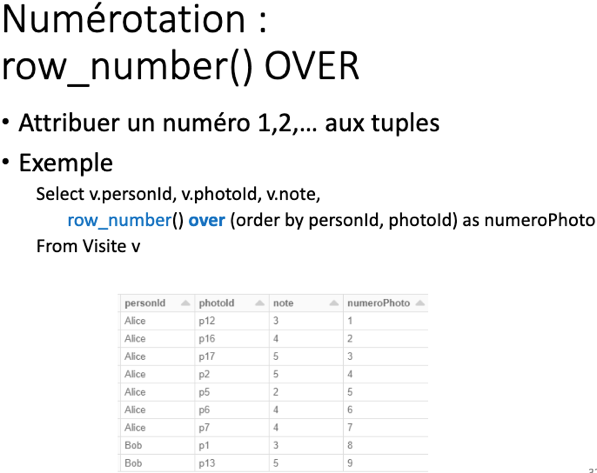
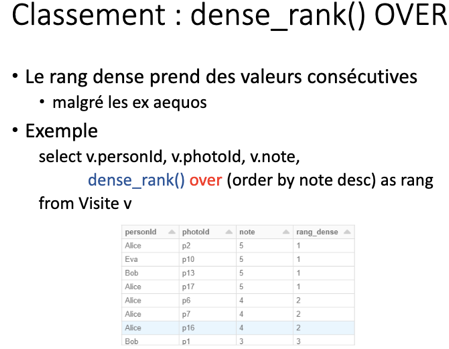
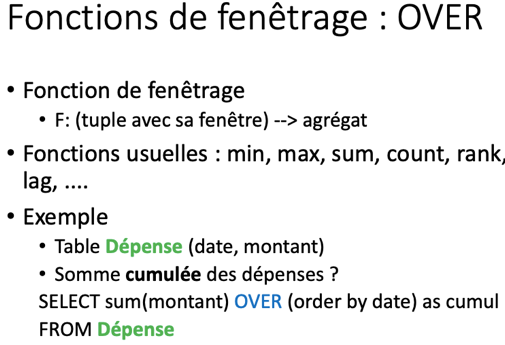
**ANALYSE MULTI DIM SQL (1)**

**  **

****

** **

**Notes projet IMDB :**

*// creation de tables ::*

**schema\_movie\_companies = """**

**id int,**

**movie\_id int,**

**company\_id int,**

**company\_type\_id int,**

**note string**

**"""**

**movie\_companies = spark.read.csv(path = dir + "movie\_companies.csv", schema = schema\_movie\_companies).persist()**

**movie\_companies.createOrReplaceTempView("Movie\_Companies")**

**create or replace temp view DeathAge as**

**select p.id, p.name, p.birth\_date, p.death\_date,**

*//remplace le jour et le mois par du vide pou recup l’année et faire la soustraction de l’age*

**regexp\_replace(death\_date, "[0-9]+ [a-zA-Z]+ ","") - regexp\_replace(birth\_date, "[0-9]+ [a-zA-Z]+ ","")as age\_of\_death,**

//fait la separation entre deux mots, séparé par un ( pour obtenir [mot1,mot2])

**split(death\_notes, "\\(") as death\_info**

**from PersonDeathGlobal p**

**;**

**create or replace temp view DeathCause as**

**select p.id, p.name, p.birth\_date, p.death\_date, p.age\_of\_death,**

**element\_at(p.death\_info,1) as death\_place,** //récupère l’élément à la place 1

**replace(element\_at(p.death\_info,2),")","") as death\_cause** //récupère l’élément à la place 2

**from DeathAge p**

**;**

**create or replace temporary view Date as**

**select distinct d.release\_date,**

**regexp\_replace(d.release\_date,"[a-zA-Z]\*( )\*([0-9]{4})\*","") as day,**

**regexp\_replace(d.release\_date,"[0-9]| ","") as month,**

**regexp\_replace(d.release\_date,"[0-9]\* [a-zA-Z]\*|[a-zA-Z]\*","") as year**

**from CastFromMovie d;**

**create or replace temporary view DateSplit as**

**select distinct d.release\_date,**

**nullif(d.day,'') as day,** *//met la ligne à null si ya pas de val*

**nullif(d.month,'') as month,**

**nullif(d.year,'') as year**

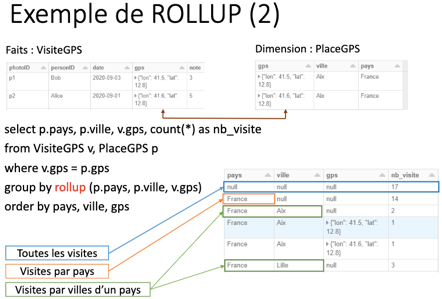
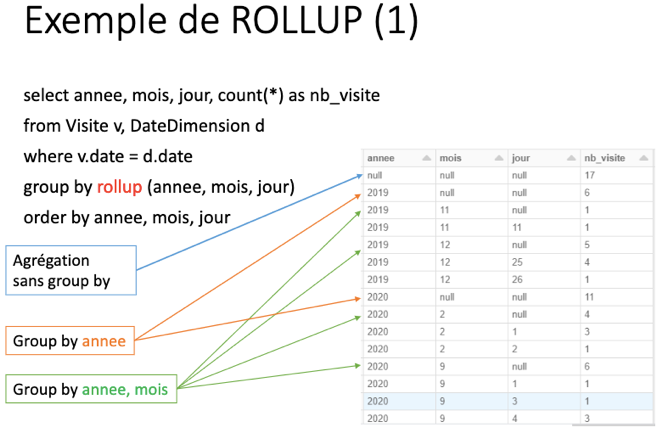
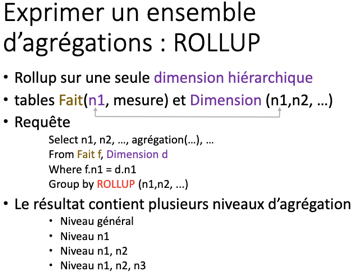
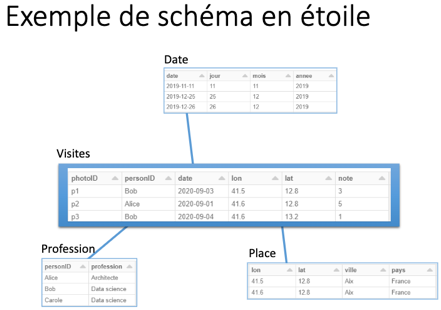
**from Date d;**

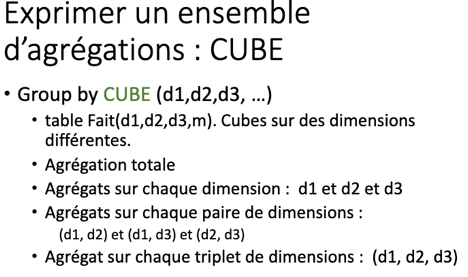
**select distinct year, month, day, count(\*) as nb\_release from CastFromMovie v, DateSplit d**

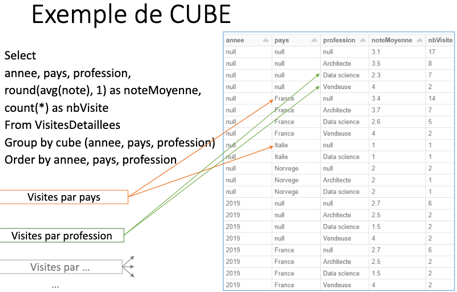
**where v.release\_date = d.release\_date**

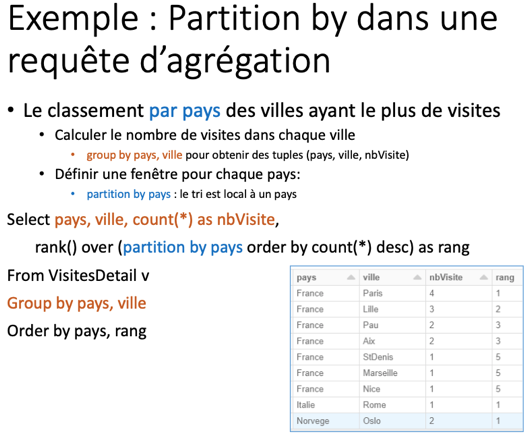
**group by rollup (year, month, day)**

**order by year, month, day**

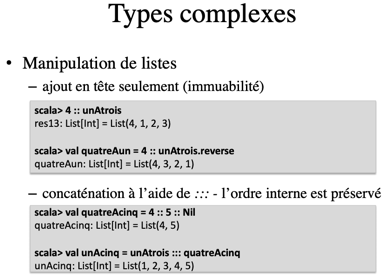
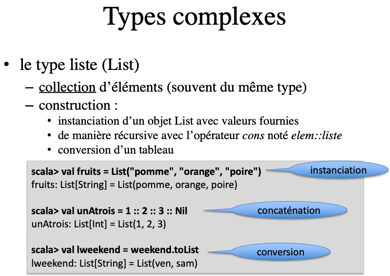
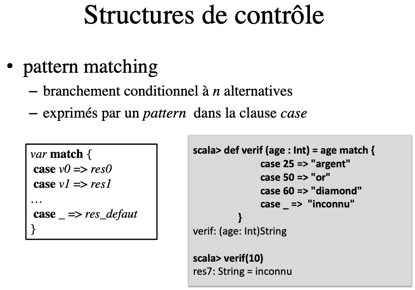
****

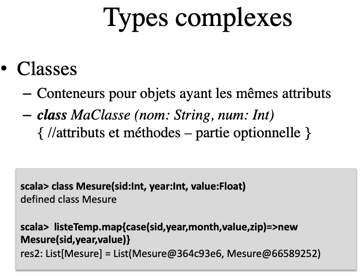
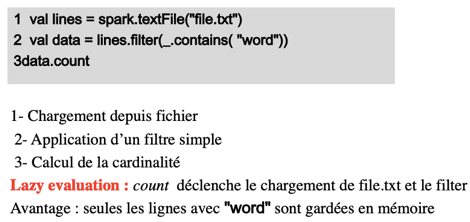
****

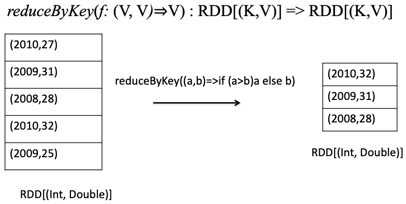
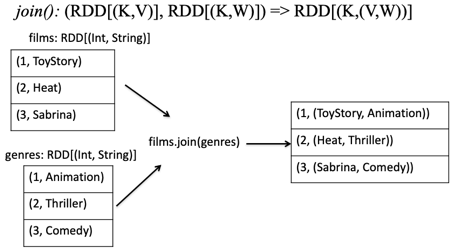
****

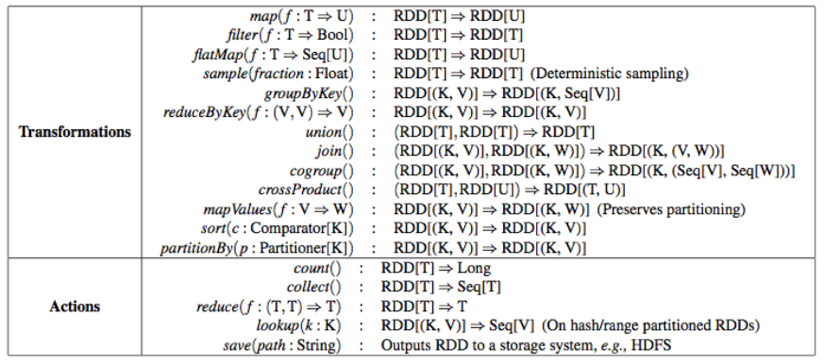
****

**SCALA SPARK (4)**

****

** **

** **

****

--------- TP : scala spark RDD ---------

**Opération sur List => .lenght | .max | .sum | .ranges**

**case class Etu(nom:String, annee:Int)**

**case class Ens(nom:String, annee:Int)**

**val classes\_personnes = personnes.map{x => x match{**

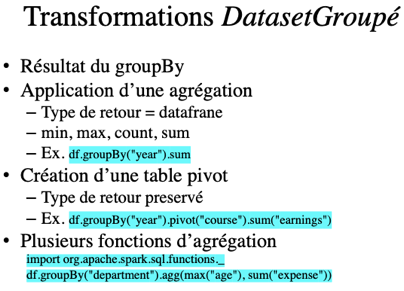
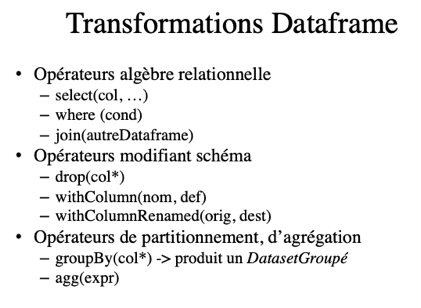
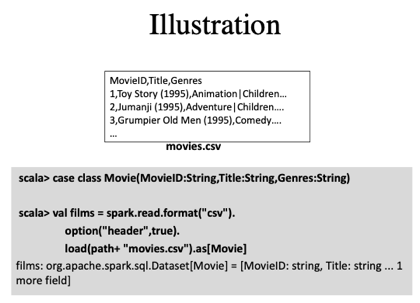
**case (nom,"etu",a) => Etu(nom,a)** *// new si c’était pas des case*

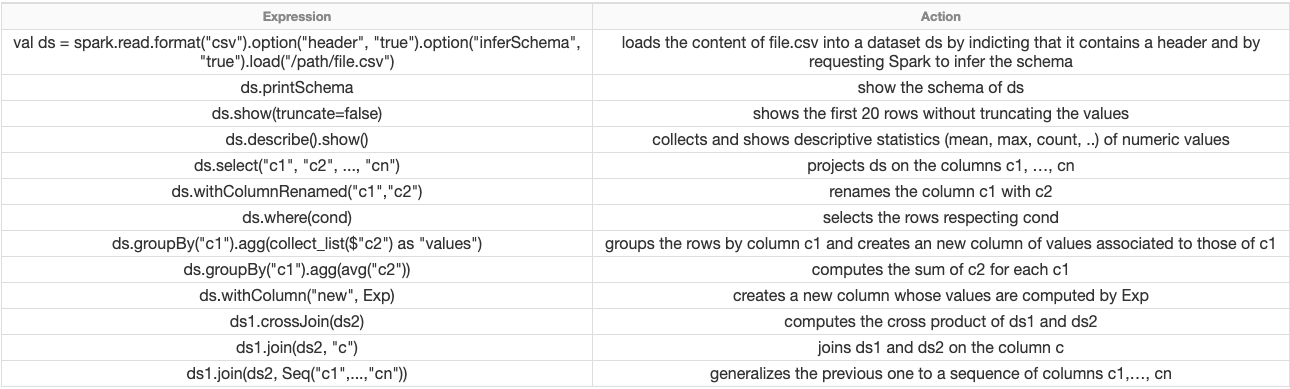
**case(nom,"ens",a) => Ens(nom,a)**

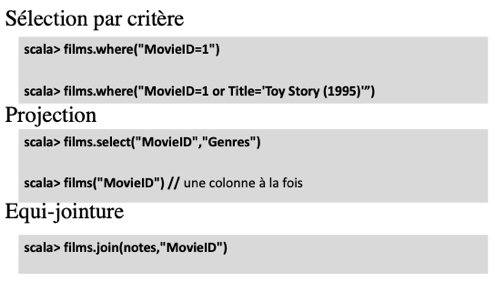
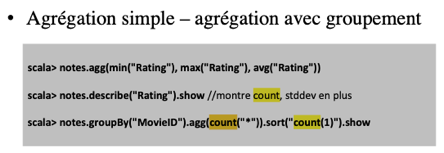
**case \_ => null**

**}**

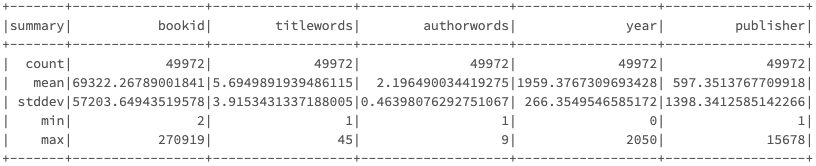
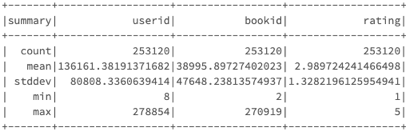
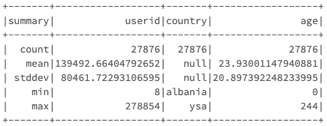
**SPARK SQL (5)**

****

****

****

--------- TP : dataframe et structure de données ---------

****

#### val s0 = users\_df.select("userid").where("country='france'") *//Identifiants des utilisateurs du pays 'france'*

#### val s1 = books\_df.select("bookid").where("year='2000'") *//Identifiants des livres publiés en 2000*

#### val q1 = users\_df.groupBy("country").count().sort($"count".desc) *//Nombre d'utilisateurs par pays avec tri décroissant*

#### *//Année avec le plus grand nombre de livres édités. Il n y a pas d'ex aequo*

#### val q3 = books\_df.groupBy("year").count().sort($"count".desc).limit(1).withColumnRenamed("count","nbLivres")

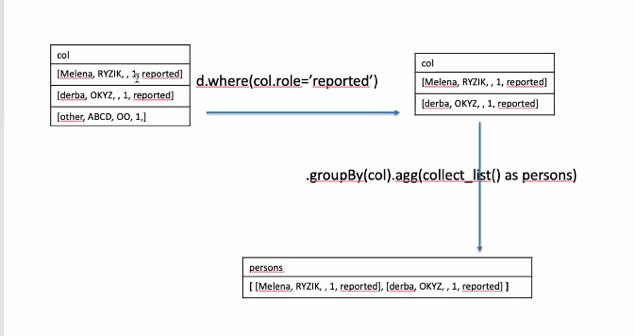
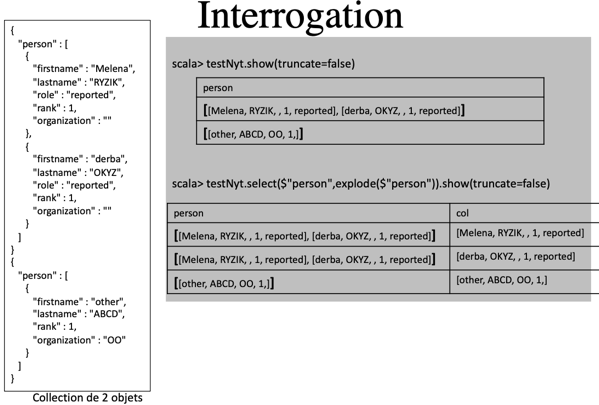
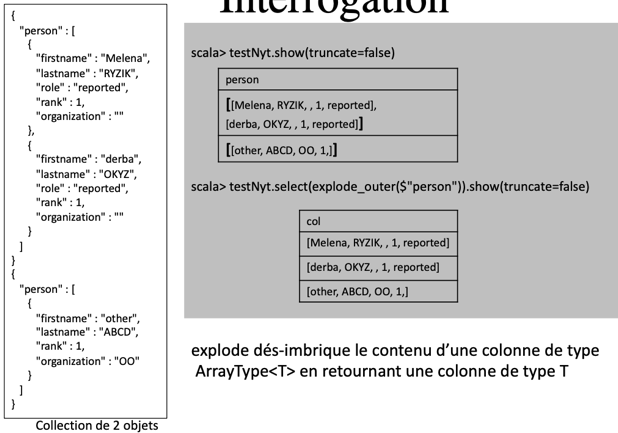
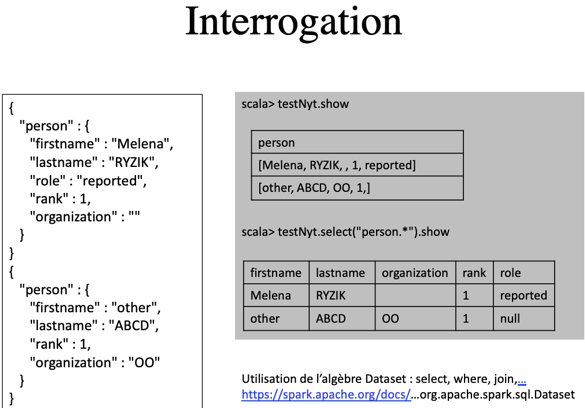
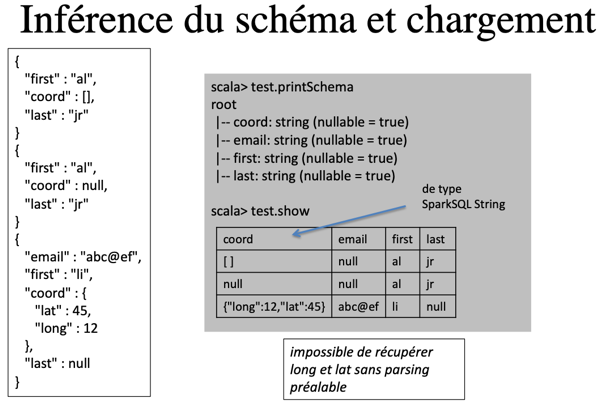
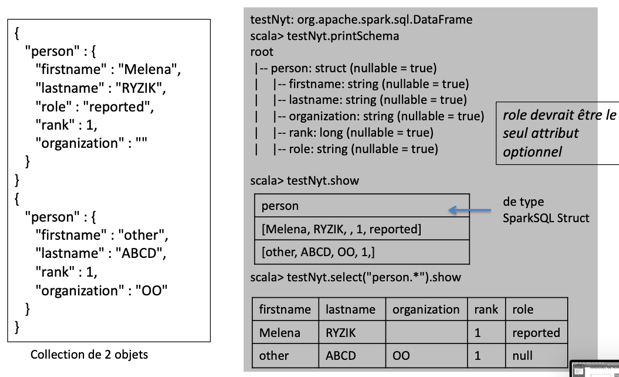
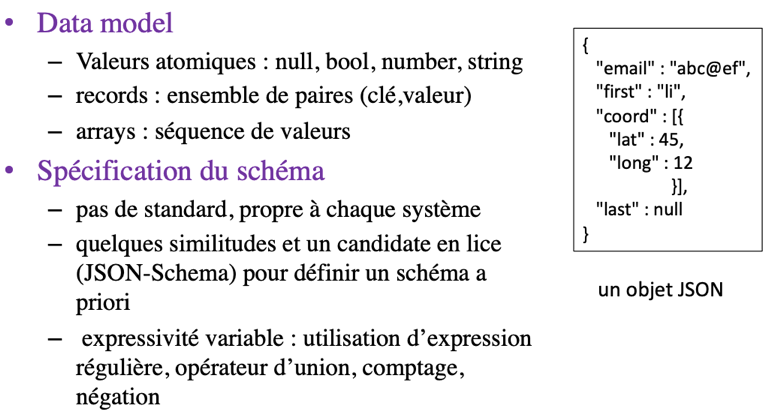
#### val q6 = ratings\_df.groupBy("bookid").agg(avg("rating") as "avg\_rating") *//La note moyenne par livre*

#### val q11 = ratings\_df.join(users\_df, "userid").groupBy("country") *//Pour chaque pays, la moyenne de livres notés par des utilisateurs de ce pays*

*//prend en entrée deux ensembles d'identifiants de livre retourne la cardinalité de l'union des deux ensembles de entrée*

**def unionBooks: UserDefinedFunction = udf((l: WrappedArray[String], r: WrappedArray[String]) => l.union(r).distinct.length.toDouble)**

**SPARK SQL SEMI STRUCT (5)**

****

--------- TP : dataframe et données semi-structuré ---------

Aggr : agregate with or without grouping. Si on veut réaliser un grouping alors faire un groupBy avant le agg

// nombre d’objet par type distinct

**val postPerType = data.groupBy("event.event\_type").agg(countDistinct("event.event\_id.post\_id").as("count"))**

*//Applatissement des listes de tags L'instruction suivante applatie ce tableau en créant une ligne pour chaque chaine contenue dans tags.*

**val dataWithTags = data.withColumn("tag", explode($"event.tags"))**

*//Retourner, pour chaque tag, le nombre d'auteurs distincts, triés de façon décroissante*

**val a =dataWithTags.groupBy("tag").agg(countDistinct("event.author.id").as("count")).orderBy($"count".desc)**

#### *//Rajouter à la table de votes le rang obtenu à partir du nombre de votes et celui obtenu à partir du nombre d'auteurs.*

**val votes\_order = Window.orderBy(desc("votes"))**

**val authors\_order = Window.orderBy(desc("nbAuths"))**

**votesCount.select("name","votes","nbAuths").withColumn("votes\_rank",rank().over(votes\_order)).withColumn("nbAuth\_rank",rank().over(authors\_order)).show()**

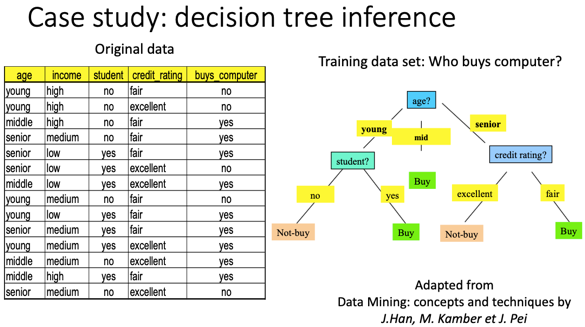
*//Rajouter un attribut month contenant le mois extrait pour chaque objet*

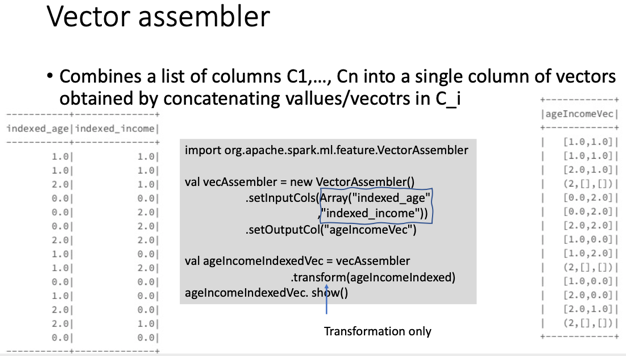
**val dataTagMon = dataWithTags.withColumn("month", month(from\_unixtime(col("event.creation\_time"))))**

*//une table pivot en réduisant le cube précédent aux dimensions mois et type d'évenement*

**val monthEvent = tag\_event\_month.groupBy("month").pivot("event\_type").sum("count").orderBy($"month")**

**SPARK ML (6)**

****

****