DEVOXX FRANCE

SparkSQL pour analyser vos données Cassandra

#TiaSparkSQL

@alexanderDeja @maxospiquante

Qui sommes-nous ?

DEVOXX FRANCE

Alexander DEJANOVSKI
@alexanderDeja
Développeur

chronopost

Maxence LECOINTE
@maxospiquante
Développeur

chronopost

Cassandra

- Base NoSQL distribuée
- Langage de requête : **CQL**~=SQL
  - *SELECT * FROM ze_table WHERE ze_key=1*
- **Pas de jointure, pas de group by, pas d'insert/select**

Spark

- Map/Reduce en mémoire
- 10x-100x plus rapide que Hadoop
- **Scala, Java ou Python**
- Modules : Spark Streaming, MLlib, GraphX, **SparkSQL**

Objectif

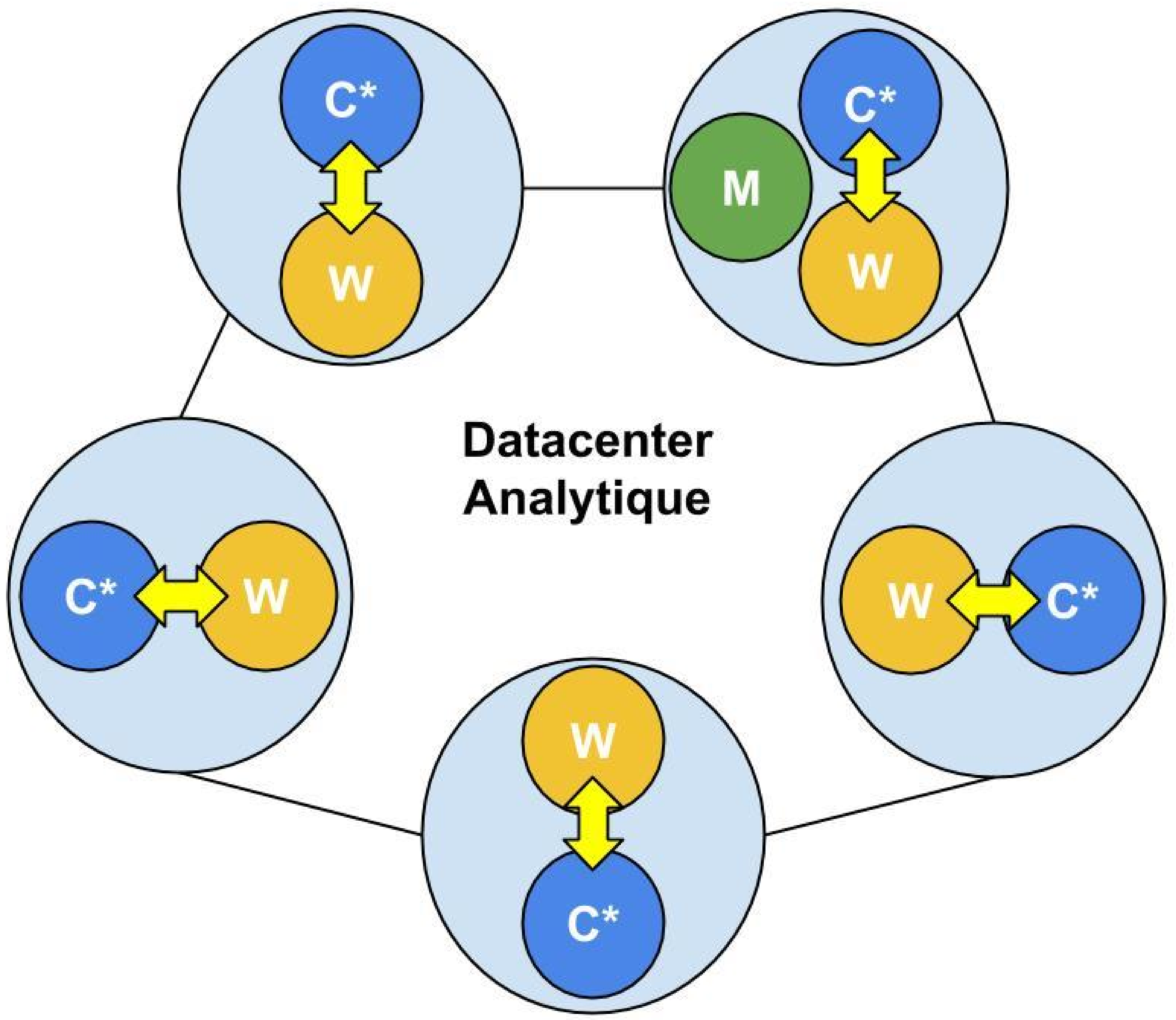
Cassandra << >> SparkSQL

Création de tables d'index

Calcul de statistiques (simples…)

sur les confs Devoxx FR de 2012 à 2015

Datastax Spark Cassandra Connector



M	= Spark Master
W	= Spark Worker
C*	= Noeud Cassandra
⇕	= Communication Spark/Cassandra via le connecteur Datastax

Setup

- Spark 1.1 ou 1.2 pour **Scala** et **Java**
  - Connecteur Datastax : http://github.com/datastax/spark-cassandra-connector
- Spark 1.1 pour **Python**
  - Connecteur Calliope de TupleJump : http://tuplejump.github.io/calliope/start-with-sql.html

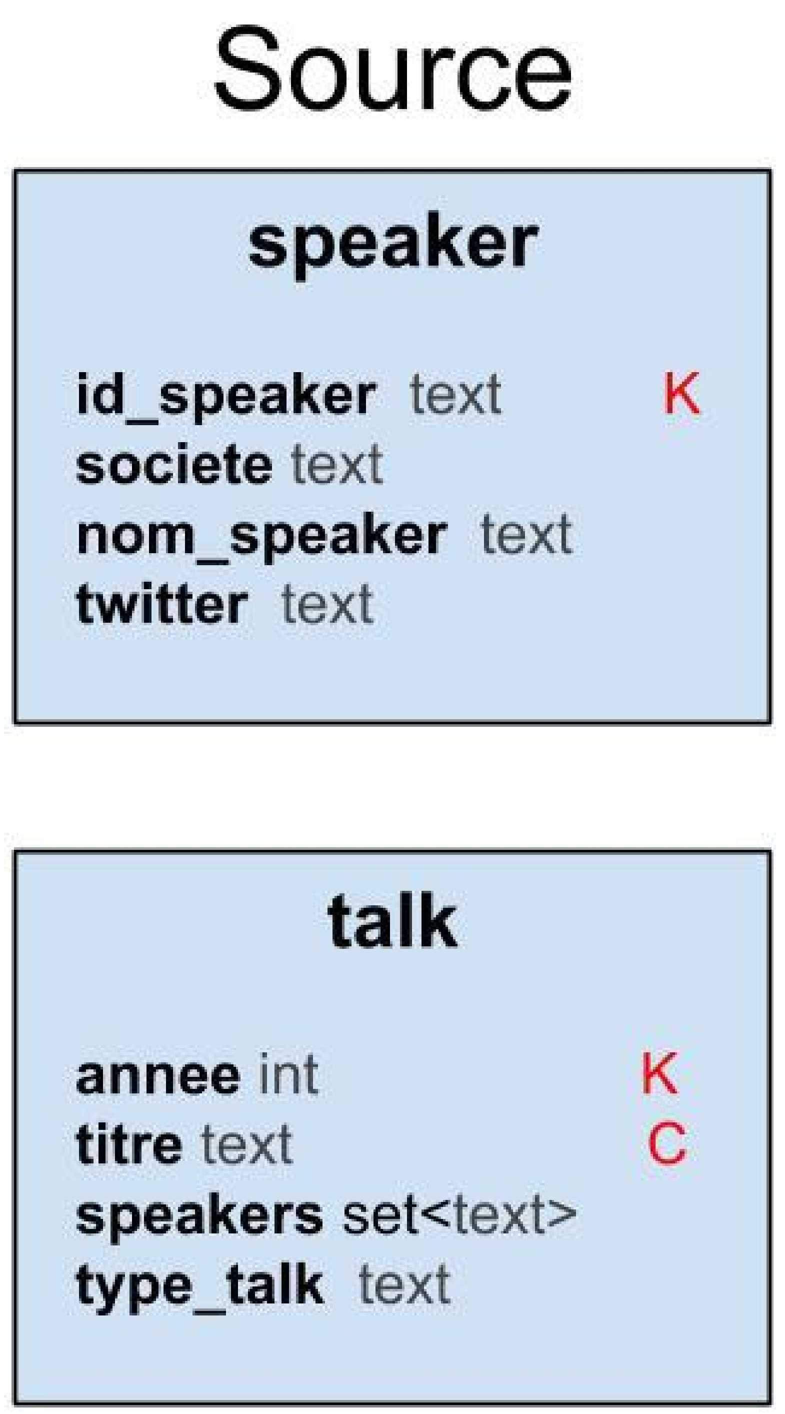
Pour vous éviter (certaines) galères…

- Sources de ce TIA : https://github.com/adejanovski/devoxx2015
- Lisez le README

C’est quoi un RDD ?

- **Resilient Distributed Dataset**
- Collection d’objet distribuée et résiliente
- Permet le stockage de n’importe quel format de donnée

Schéma


Source

speaker

id_speaker text K
societe text
nom_speaker text
twitter text

talk

annee int K
titre text C
speakers set<text>
type_talk text

Schéma

Source

speaker

id_speaker	text	K
societe	text	
nom_speaker	text	
twitter	text	

talk

annee	int	K
titre	text	C
speakers	set<text>	
type_talk	text	

Statistiques

societe_par_annee

societe	text	K
annee	int	C
nb	int	

speaker_par_annee

nom_speaker	text	K
annee	int	C
nb	int	
id_speaker	text	

keyword_par_annee

keyword	text	K
annee	int	C
nb	int	

Index

talk_par_speaker

id_speaker	text	K
type_talk	text	C
titre	text	C
annee	text	

talk_par_societe

societe	text	K
type_talk	text	C
titre	text	C
annee	text	

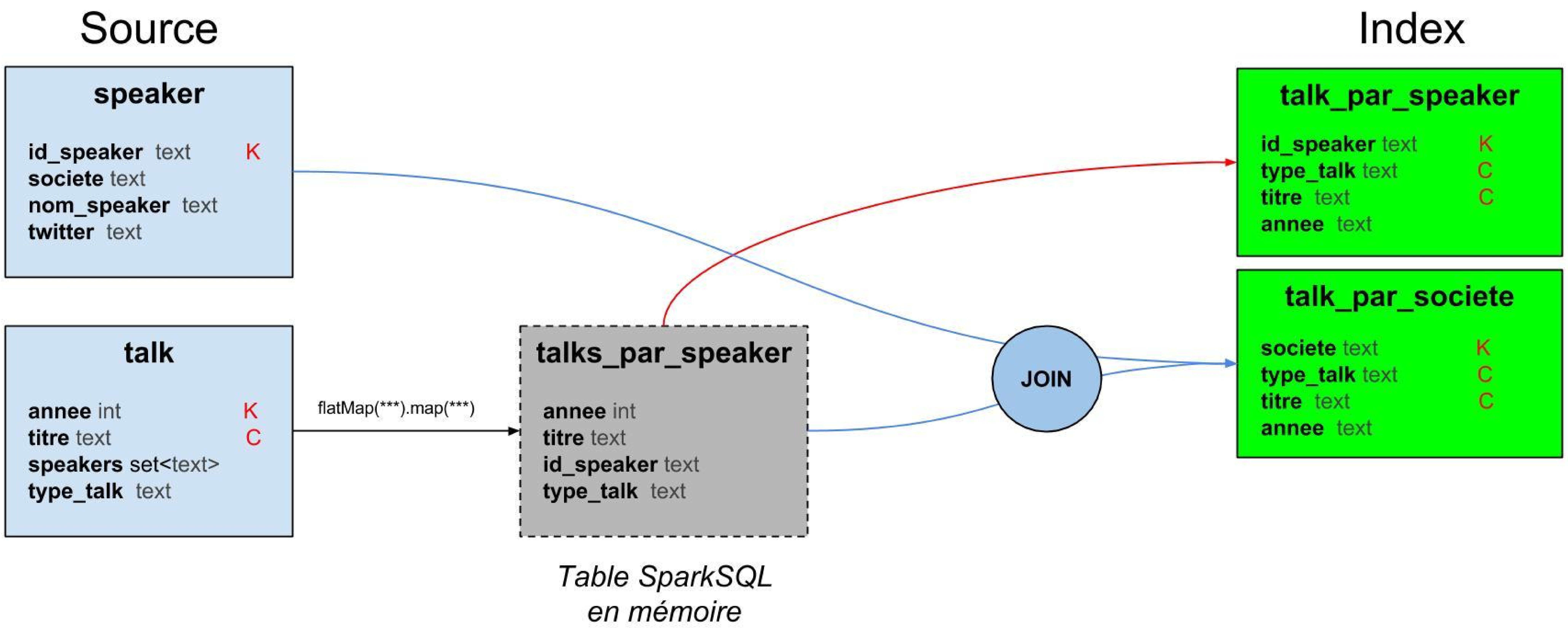
speaker_par_societe

societe	text	K
id_speaker	text	C
nom_speaker	text	

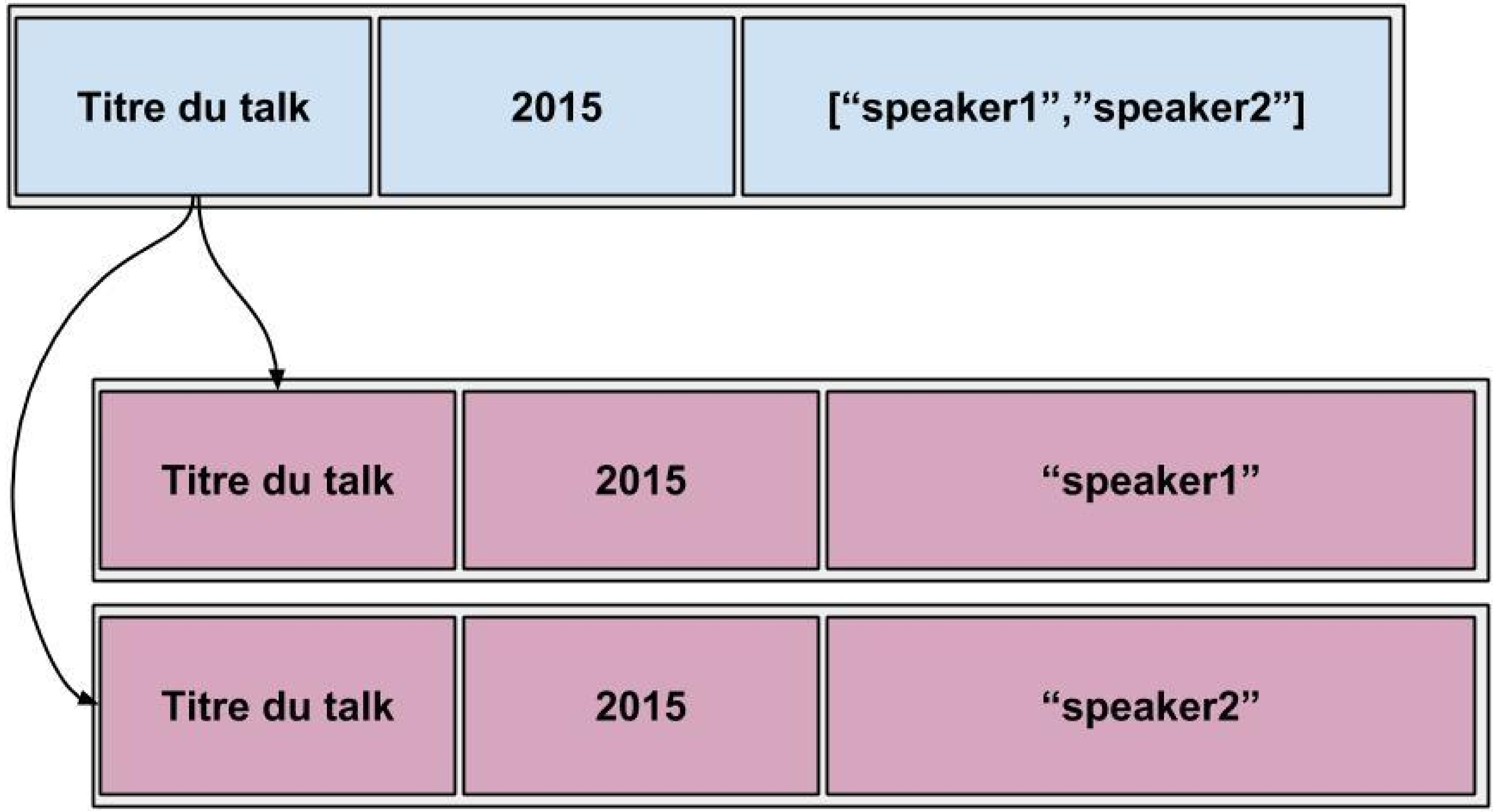
Etape 1

Scala

Scala-Fu


Source
Index
speaker
id_speaker text K
societe text
nom_speaker text
twitter text
talk
annee int K
titre text C
speakers set<text>
type_talk text
flatMap(***).map(***)
talks_par_speaker
annee int
titre text
id_speaker text
type_talk text
Table SparkSQL en mémoire
JOIN
talk_par_speaker
id_speaker text K
type_talk text C
titre text C
annee text
talk_par_societe
societe text K
type_talk text C
titre text C
annee text

Scala-Fu : split par speaker



Code Scala

```
val rddTalk = cc.sql("select annee, titre, speakers, type_talk
                      from devoxx.talk")

// On sort de SparkSQL pour retravailler les données
val splitBySpeakersRdd =
rddTalk.flatMap(r => (r(2).asInstanceOf[scala.collection.immutable.Set[String]])
     .map(m => (m,r) ))

case class Talk(titre: String, speaker: String, annee: Int, type_talk: String)
val talksSchemaRdd = splitBySpeakersRdd.map(
  t =>Talk(t._2.getString(1),t._1,t._2.getInt(0),t._2.getString(1),t._2.getString(3)))

talksSchemaRdd.registerTempTable("talks_par_speaker")
```

Code Scala

```
val rddTalk = cc.sql("select annee, titre, speakers, type_talk
                                from devoxx.talk")

// On sort de SparkSQL pour retravailler les données
val splitBySpeakersRdd =
rddTalk.flatMap(r => (r(2).asInstanceOf[scala.collection.immutable.Set[String]])
     .map(m => (m,r) ))

case class Talk(titre: String, speaker: String, annee: Int, type_talk: String)
val talksSchemaRdd = splitBySpeakersRdd.map(
  t =>Talk(t._2.getString(1),t._1,t._2.getInt(0),t._2.getString(1),t._2.getString(3)))

talksSchemaRdd.registerTempTable("talks_par_speaker")
```

Code Scala

```
val rddTalk = cc.sql("select annee, titre, speakers, type_talk
                           from devoxx.talk")

// On sort de SparkSQL pour retravailler les données
val splitBySpeakersRdd =
rddTalk.flatMap(r => (r(2).asInstanceOf[scala.collection.immutable.Set[String]])
    .map(m => (m,r) ))

case class Talk(titre: String, speaker: String, annee: Int, type_talk: String)
val talksSchemaRdd = splitBySpeakersRdd.map(
  t =>Talk(t._2.getString(1),t._1,t._2.getInt(0),t._2.getString(1),t._2.getString(3)))

talksSchemaRdd.registerTempTable("talks_par_speaker")
```

Code Scala

```
val rddTalk = cc.sql("select annee, titre, speakers, type_talk
                          from devoxx.talk")

// On sort de SparkSQL pour retravailler les données
val splitBySpeakersRdd =
rddTalk.flatMap(r => (r(2).asInstanceOf[scala.collection.immutable.Set[String]])
    .map(m => (m,r) ))

case class Talk(titre: String, speaker: String, annee: Int, type_talk: String)
val talksSchemaRdd = splitBySpeakersRdd.map(
  t =>Talk(t._2.getString(1),t._1,t._2.getInt(0),t._2.getString(1),t._2.getString(3)))

talksSchemaRdd.registerTempTable("talks_par_speaker")
```

Code Scala : insertion Cassandra

```
cc.sql("insert into devoxx.talk_par_speaker
        select speaker, type_talk, titre, annee
        from talks_par_speaker").collect()
```

Code Scala : insertion Cassandra

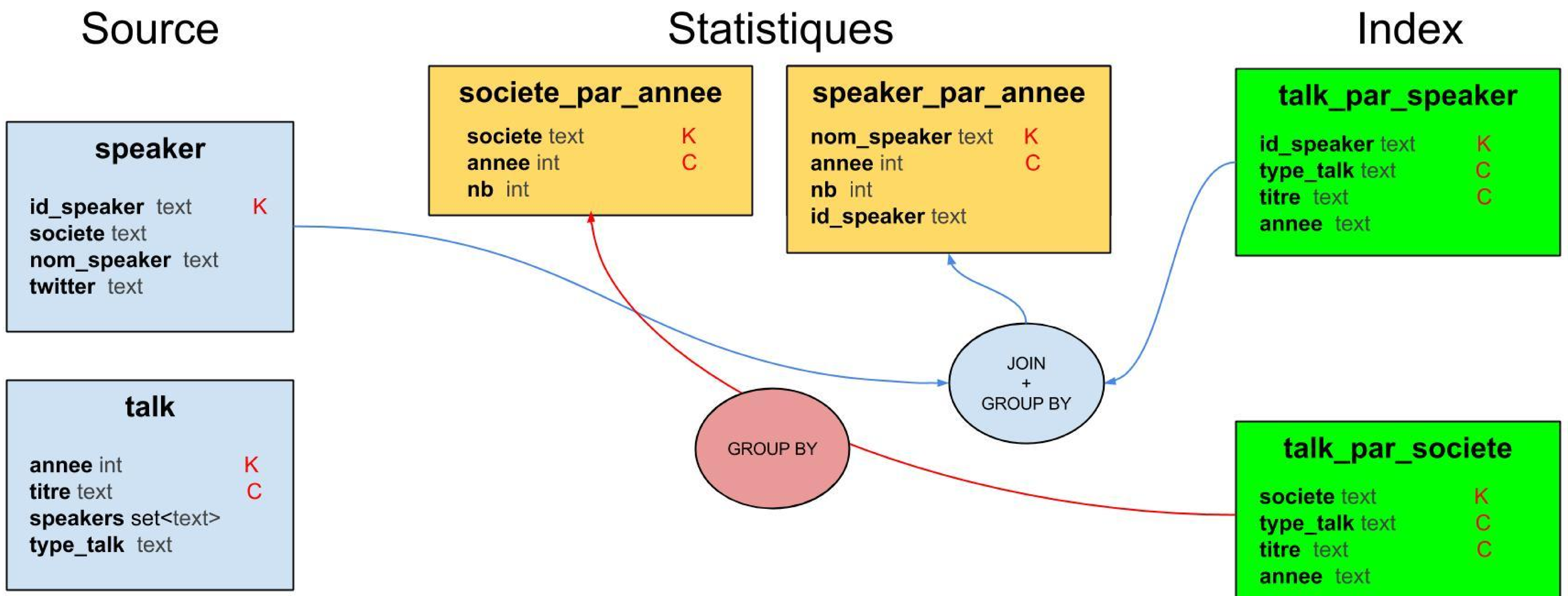
```
val connector = CassandraConnector(sc.getConf)

talksSchemaRdd.foreachPartition(partition => {
    connector.withSessionDo{ session =>
       partition.foreach(r => session.execute(
                  "UDPATE devoxx.talk_par_speaker USING TTL ? " +
                  set type_talk=?, titre=?, annee=? " +
                  WHERE id_speaker = ?"),
                  86400, r.type_talk, r.titre,
                  r.annee.asInstanceOf[java.lang.Integer],r.speaker)
       )}
})
```

“Demo time”

Etape 2

Java™

Java-Fu



Source
Statistiques
Index
speaker
id_speaker text K
societe text
nom_speaker text
twitter text
talk
annee int K
titre text C
speakers set<text>
type_talk text
societe_par_annee
societe text K
annee int C
nb int
speaker_par_annee
nom_speaker text K
annee int C
nb int
id_speaker text
talk_par_speaker
id_speaker text K
type_talk text C
titre text C
annee text
talk_par_societe
societe text K
type_talk text C
titre text C
annee text
JOIN + GROUP BY
GROUP BY

Code Java

```
SchemaRDD nbTalkParSpeaker = cassandraSQLContext.sql(
        “SELECT B.nom_speaker as nom_speaker, A.annee as annee,
        “A.id_speaker as id_speaker “ +
        “FROM devoxx.talk_par_speaker A JOIN  devoxx.speakers B “+
        “ON A.id_speaker = B.id_speaker ");
nbTalkParSpeaker.registerTempTable(“tmp_talk_par_speaker");
cassandraSQLContext.sql(
        “INSERT INTO devoxx.speaker_par_annee “ +
        “SELECT nom_speaker, annee, count(*) as nb “+
        “FROM tmp_talk_par_speaker group by nom_speaker, annee").collect();
```

Code Java

```
SchemaRDD nbTalkParSpeaker = cassandraSQLContext.sql(
        “SELECT B.nom_speaker as nom_speaker, A.annee as annee,
        “A.id_speaker as id_speaker “ +
        “FROM devoxx.talk_par_speaker A JOIN  devoxx.speakers B “+
        “ON A.id_speaker = B.id_speaker ");

nbTalkParSpeaker.registerTempTable(“tmp_talk_par_speaker");

cassandraSQLContext.sql(
        “INSERT INTO devoxx.speaker_par_annee “ +
        “SELECT nom_speaker, annee, count(*) as nb “+
        “FROM tmp_talk_par_speaker group by nom_speaker, annee").collect();
```

Code Java

```
SchemaRDD nbTalkParSpeaker = cassandraSQLContext.sql(
        "SELECT B.nom_speaker as nom_speaker, A.annee as annee,
        "A.id_speaker as id_speaker " +
        "FROM devoxx.talk_par_speaker A JOIN  devoxx.speakers B "+
        "ON A.id_speaker = B.id_speaker ");

nbTalkParSpeaker.registerTempTable("tmp_talk_par_speaker");

cassandraSQLContext.sql(
        "INSERT INTO devoxx.speaker_par_annee " +
        "SELECT nom_speaker, annee, count(*) as nb, id_speaker "+
        "FROM tmp_talk_par_speaker "+
        "GROUP BY nom_speaker, annee, id_speaker").collect();
```

Submit Java

```
./spark-submit                                    \
        --class devoxx.Devoxx…..                  \
        --master spark://127.0.0.1:7077           \
        devoxxSparkSql.jar
```

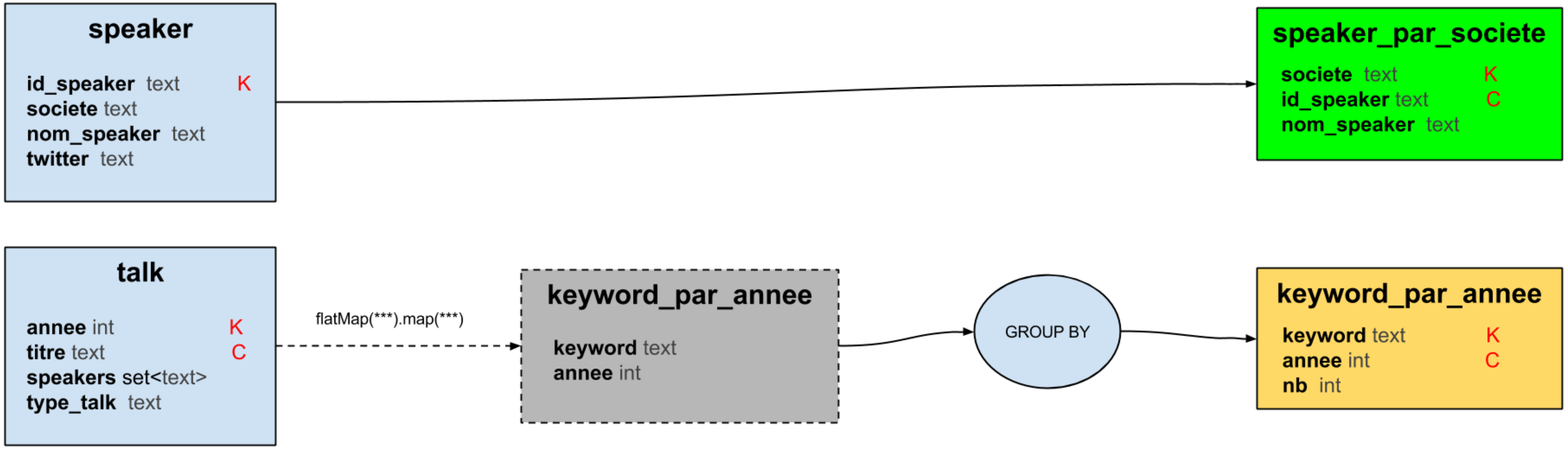
“Demo time”

Etape 3

python

Python-Fu

Source



Code Python

```
def split_keywords(row):
      ## fonction splittant les titres par mot

rddTalk = sqlContext.sql("SELECT titre, speakers, annee, categorie, type_talk FROM devoxx.talk")
splitByKeywordRdd = rddTalk.flatMap(lambda r:split_keywords(r))
splitByKeywordRdd_schema = sqlContext.inferSchema(
                           splitByKeywordRdd.filter(lambda word:len(word[0])>1)
                                       .map(lambda x:Row(keyword=x[0],annee=x[1])))
splitByKeywordRdd_schema.registerTempTable("tmp_keywords")
keyword_count = sqlContext.sql("""SELECT keyword, annee, count(*) as nb
                           FROM tmp_keywords
                           GROUP BY keyword, annee""")
keyword_count_schema = sqlContext.inferSchema(keyword_count.map(lambda x:Row(...)))
keyword_count_schema.registerTempTable("tmp_keywords_count")
sqlContext.sql("""INSERT INTO devoxx.keyword_par_annee SELECT keyword, annee, nb
            FROM tmp_keywords_count""")
```

Code Python

```
def split_keywords(row):
      ## fonction splittant les titres par mot

rddTalk = sqlContext.sql("select titre, speakers, annee, categorie, type_talk from devoxx.talk")
splitByKeywordRdd = rddTalk.flatMap(lambda r:split_keywords(r))
splitByKeywordRdd_schema = sqlContext.inferSchema(
                                        splitByKeywordRdd.filter(lambda word:len(word[0])>1)
                                                         .map(lambda x:Row(keyword=x[0],annee=x[1])))
splitByKeywordRdd_schema.registerTempTable("tmp_keywords")
keyword_count = sqlContext.sql("""SELECT keyword, annee, count(*) as nb
                                        FROM tmp_keywords
                                        GROUP BY keyword, annee""")
keyword_count_schema = sqlContext.inferSchema(keyword_count.map(lambda x:Row(...)))
keyword_count_schema.registerTempTable("tmp_keywords_count")
sqlContext.sql("""INSERT INTO devoxx.keyword_par_annee SELECT keyword, annee, nb
                    FROM tmp_keywords_count""")
```

Code Python

```
def split_keywords(row):
      ## fonction splittant les titres par mot

rddTalk = sqlContext.sql("select titre, speakers, annee, categorie, type_talk from devoxx.talk")
splitByKeywordRdd = rddTalk.flatMap(lambda r:split_keywords(r))
splitByKeywordRdd_schema = sqlContext.inferSchema(
                                          splitByKeywordRdd.filter(lambda word:len(word[0])>1)
                                                                .map(lambda x:Row(keyword=x[0],annee=x[1])))
splitByKeywordRdd_schema.registerTempTable("tmp_keywords")
keyword_count = sqlContext.sql("""SELECT keyword, annee, count(*) as nb
                                          FROM tmp_keywords
                                          GROUP BY keyword, annee""")
keyword_count_schema = sqlContext.inferSchema(keyword_count.map(lambda x:Row(...)))
keyword_count_schema.registerTempTable("tmp_keywords_count")
sqlContext.sql("""INSERT INTO devoxx.keyword_par_annee SELECT keyword, annee, nb
                    FROM tmp_keywords_count""")
```

Code Python

```
def split_keywords(row):
      ## fonction splittant les titres par mot

rddTalk = sqlContext.sql("select titre, speakers, annee, categorie, type_talk from devoxx.talk")
splitByKeywordRdd = rddTalk.flatMap(lambda r:split_keywords(r))
splitByKeywordRdd_schema = sqlContext.inferSchema(
                                    splitByKeywordRdd.filter(lambda word:len(word[0])>1)
                                                   .map(lambda x:Row(keyword=x[0],annee=x[1])))
splitByKeywordRdd_schema.registerTempTable("tmp_keywords")
keyword_count = sqlContext.sql("""SELECT keyword, annee, count(*) as nb
                                    FROM tmp_keywords
                                    GROUP BY keyword, annee""")
keyword_count_schema = sqlContext.inferSchema(keyword_count.map(lambda x:Row(...)))
keyword_count_schema.registerTempTable("tmp_keywords_count")
sqlContext.sql("""INSERT INTO devoxx.keyword_par_annee SELECT keyword, annee, nb
               FROM tmp_keywords_count""")
```

“Demo time”

Et voilà ! **Des questions ?**