Traitement de données semi-structurées à large échelle

Master DAC – Bases de Données Large Echelle

Mohamed-Amine Baazizi
baazizi@ia.lip6.fr

Novembre 2018

Contexte

- Données semi-structurées
  - très répandues : crawl api, datasets publiques, …
  - flexibilité, pas de schema pré-établi (ou a posteriori)
  - imbrication sur plusieurs niveaux, structure variable
  - difficiles à manipuler et à cerner
- JSON : modèle le plus répondu
  - syntaxe plus simple de XML
  - exprime à la fois une séquence de tuples ou de valeurs (avec un array) et un ensemble de tuples (avec un record)
- quelques extensions utilisent
  - Map (tableaux associatifs)
  - Bags (arrays sans la notion d'ordre)

JSON : modèle de données et langage de schémas

- Data model
  - Valeurs atomiques : null, bool, number, string
  - records : ensemble de paires (clé,valeur)
  - arrays : séquence de valeurs
- Schema language
  - pas de standard, propre à chaque système
  - quelques similitudes et un candidate en lice (JSON-Schema) pour définir un schéma a priori
  - expressivité variable : utilisation d'expression régulière, opérateur d'union, comptage, négation

```
{
  "email" : "abc@ef",
  "first" : "li",
  "coord" : [{
    "lat" : 45,
    "long" : 12
             }],
  "last" : null
}
```

un objet JSON

Cas Spark

- Langage de schéma
  - types de base : boolean, numeric (integer, decimal, …), String, null, timestamp
  - type tableau : ArrayType(element, containsNull)
  - type object : StructType(List [StructField])
    - StructField(name, type, nullable)
  - type tableau associatif : MapType (keyType, valueType, valueContainsNull)
- Inférence du schéma pour une collection
  - deux étapes : inférer le type de chaque objet puis fusionner les types
  - tous les attributs marqués comme optionels (nullable=true)
  - inférer String lorsque la collection est hétérogène (différentes sortes de types)
- Stockage des données : relationnel étendu
  - types imbriqués SparkSQL : Objet, Tableau, Tableau Associatif
  - Génération de tables SparkSQL guidée par le schéma
  - Schéma automatiquement inféré lorsqu'il n'est pas fourni
  - utilisation des null pour indiquer l'absence d'attributs
- Interrogation semblable à l'objet-relationnel
  - algèbre Dataset + navigation dans la structure imbriquée à l'aide de la notation pointée
  - dés-imbrication (unnest) ou groupement (nest)
  - analyse statique pour inférer le type de chaque expression

Inférence du schéma et Stockage

```
{
  "person" : {
    "firstname" : "Melena",
    "lastname" : "RYZIK",
    "role" : "reported",
    "rank" : 1,
    "organization" : ""
  }
}
{
  "person" : {
    "firstname" : "other",
    "lastname" : "ABCD",
    "rank" : 1,
    "organization" : "OO"
  }
}
```

Collection de 2 objets

```
testNyt: org.apache.spark.sql.DataFrame
scala> testNyt.printSchema
root
 |-- person: struct (nullable = true)
 |    |-- firstname: string (nullable = true)
 |    |-- lastname: string (nullable = true)
 |    |-- organization: string (nullable = true)
 |    |-- rank: long (nullable = true)
 |    |-- role: string (nullable = true)
```

role devrait être le seul attribut optionnel

```
scala> testNyt.show
```

person
[Melena, RYZIK, , 1, reported]
[other, ABCD, OO, 1,]

de type SparkSQL Struct

```
scala> testNyt.select("person.*").show
```

firstname	lastname	organization	rank	role
Melena	RYZIK		1	reported
other	ABCD	OO	1	null

Inférence du schéma et Stockage

```
{
  "first" : "al",
  "coord" : [],
  "last" : "jr"
}
{
  "first" : "al",
  "coord" : null,
  "last" : "jr"
}
{
  "email" : "abc@ef",
  "first" : "li",
  "coord" : {
    "lat" : 45,
    "long" : 12
  },
  "last" : null
}
```

Collection de 3 objets

```
scala> test.printSchema
root
 |-- coord: string (nullable = true)
 |-- email: string (nullable = true)
 |-- first: string (nullable = true)
 |-- last: string (nullable = true)

scala> test.show
```

de type SparkSQL String

coord	email	first	last
[ ]	null	al	jr
null	null	al	jr
{"long":12,"lat":45}	abc@ef	li	null

impossible de récupérer long et lat sans parsing préalable

Inférence du schéma et Stockage

```
{
  "person" : [
    {
      "firstname" : "Melena",
      "lastname" : "RYZIK",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    },
    {
      "firstname" : "derba",
      "lastname" : "OKYZ",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    }
  ]
}
{
  "person" : [
    {
      "firstname" : "other",
      "lastname" : "ABCD",
      "rank" : 1,
      "organization" : "OO"
    }
  ]
}
```

Collection de 2 objets

```
scala> testNyt.printSchema
root
 |-- person: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- firstname: string (nullable = true)
 |    |    |-- lastname: string (nullable = true)
 |    |    |-- organization: string (nullable = true)
 |    |    |-- rank: long (nullable = true)
 |    |    |-- role: string (nullable = true)


scala> testNyt.show(truncate=false)
```

de type
SparkSQL Array<Struct>

person
[[Melena, RYZIK, , 1, reported], [derba, OKYZ, , 1, reported]]
[[other, ABCD, OO, 1,]]

Interrogation

```
{
  "person" : {
    "firstname" : "Melena",
    "lastname" : "RYZIK",
    "role" : "reported",
    "rank" : 1,
    "organization" : ""
  }
}
{
  "person" : {
    "firstname" : "other",
    "lastname" : "ABCD",
    "rank" : 1,
    "organization" : "OO"
  }
}
```

Collection de 2 objets

scala> testNyt.show

person
[Melena, RYZIK, , 1, reported]
[other, ABCD, OO, 1,]

scala> testNyt.select("person.*").show

firstname	lastname	organization	rank	role
Melena	RYZIK		1	reported
other	ABCD	OO	1	null

Utilisation de l'algèbre Dataset : select, where, join,…
https://spark.apache.org/docs/…org.apache.spark.sql.Dataset

Interrogation

```
{
  "person" : [
    {
      "firstname" : "Melena",
      "lastname" : "RYZIK",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    },
    {
      "firstname" : "derba",
      "lastname" : "OKYZ",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    }
  ]
}
{
  "person" : [
    {
      "firstname" : "other",
      "lastname" : "ABCD",
      "rank" : 1,
      "organization" : "OO"
    }
  ]
}
```

Collection de 2 objets

scala> testNyt.show(truncate=false)

person
[[Melena, RYZIK, , 1, reported], [derba, OKYZ, , 1, reported]]
[[other, ABCD, OO, 1,]]

scala> testNyt.select(explode_outer($"person")).show(truncate=false)

col
[Melena, RYZIK, , 1, reported]
[derba, OKYZ, , 1, reported]
[other, ABCD, OO, 1,]

explode dés-imbrique le contenu d'une colonne de type ArrayType<T> en retournant une colonne de type T

Interrogation

```
{
  "person" : [
    {
      "firstname" : "Melena",
      "lastname" : "RYZIK",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    },
    {
      "firstname" : "derba",
      "lastname" : "OKYZ",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    }
  ]
}
{
  "person" : [
    {
      "firstname" : "other",
      "lastname" : "ABCD",
      "rank" : 1,
      "organization" : "OO"
    }
  ]
}
```

Collection de 2 objets

scala> testNyt.show(truncate=false)

person
[[Melena, RYZIK, , 1, reported], [derba, OKYZ, , 1, reported]]
[[other, ABCD, OO, 1,]]

scala> testNyt.select($"person",explode($"person")).show(truncate=false)

person	col
[[Melena, RYZIK, , 1, reported], [derba, OKYZ, , 1, reported]]	[Melena, RYZIK, , 1, reported]
[[Melena, RYZIK, , 1, reported], [derba, OKYZ, , 1, reported]]	[derba, OKYZ, , 1, reported]
[[other, ABCD, OO, 1,]]	[other, ABCD, OO, 1,]

Interrogation


```
{
  "person" : [
    {
      "firstname" : "Melena",
      "lastname" : "RYZIK",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    },
    {
      "firstname" : "derba",
      "lastname" : "OKYZ",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    }
  ]
}
{
  "person" : [
    {
      "firstname" : "other",
      "lastname" : "ABCD",
      "rank" : 1,
      "organization" : "OO"
    }
  ]
}
```


Collection de 2 objets

?

Nécessite l'imbrication dans le select


```
{
  "role" : "reported",
  "persons" : [
    {
      "firstname" : "Melena",
      "lastname" : "RYZIK",
      "rank" : 1,
      "organization" : ""
    },
    {
      "firstname" : "derba",
      "lastname" : "OKYZ",
      "rank" : 1,
      "organization" : ""
    }
  ]
}
```

Bilan

- Expressivité limitée du langage de schéma et du langage de requêtes
  - pas de distinction entre attributs optionnels et obligatoires
  - n'exprime pas l'union : String + [ ] + {long: String, lat: String}
  - quid de l'accès indexé aux éléments d'un Array et de l'imbrication dans le select?
- Autres pistes
  - Extensions SQL : *SQL++* de AsterixDB, *N1QL* de Couchbase, *SQL* de Apache Drill
  - Langages propriétaires : *Aggregation Pipelines* de Mongo, *JSONiq* de Zorba
  - Plusieurs connecteurs possibles avec Spark