



# Table des matières

<b>Sommaire</b>	<b>9</b>
<b>1 Java</b>	<b>10</b>
1.1 Eclipse . . . . .	10
1.1.1 Schéma . . . . .	10
1.1.2 Outils . . . . .	10
1.1.3 Mode debug . . . . .	10
1.2 Java . . . . .	11
1.2.1 Type . . . . .	11
1.3 Héritage . . . . .	11
1.3.1 Polymorphisme . . . . .	11
1.4 Interface . . . . .	12
1.5 Enumeration . . . . .	12
1.6 Généricité . . . . .	12
1.7 Collection . . . . .	13
1.7.1 List . . . . .	13
1.7.2 Set . . . . .	13
1.7.3 Map . . . . .	13
1.8 Opération ternaire . . . . .	13
1.9 IO Stream . . . . .	14
1.10 Exceptions . . . . .	14
<b>2 SGBDR : Base de données Relationnelle</b>	<b>15</b>
2.1 Formes normales . . . . .	15
2.2 Requête SQL . . . . .	15
2.2.1 Les livraisons . . . . .	15
2.2.2 La société Gavasoft . . . . .	17
2.2.3 TP Médicaments . . . . .	17
2.3 MySQL Workbench . . . . .	18
2.3.1 Diagramme . . . . .	18
2.4 MariaDB/MySQL . . . . .	18
2.4.1 Utilisation des GRANT . . . . .	18
2.4.2 Importer/Exporter . . . . .	19
<b>3 JDBC : Java Data Base Connectivity</b>	<b>20</b>

3.1	Classes et interfaces . . . . .	20
3.1.1	java.sql.DriverManager . . . . .	20
3.1.2	java.sql.Connection . . . . .	20
3.1.3	java.sql.Statement . . . . .	20
3.1.4	java.sql.ResultSet . . . . .	21
3.1.5	Vue avec eclipse . . . . .	21
3.1.6	Configuration . . . . .	21
3.1.7	Scrapbook . . . . .	21
3.2	Séparation des couches . . . . .	21
3.2.1	DAO . . . . .	21
<b>4</b>	<b>JPA : Java Persistence API</b>	<b>23</b>
4.1	Hibernate . . . . .	23
4.1.1	hibernate.cfg.xml . . . . .	23
4.1.2	hbm.xml . . . . .	24
4.1.3	Annotations . . . . .	24
4.1.4	Méthodes de hibernate . . . . .	24
4.1.5	HQL . . . . .	25
4.2	EclipseLink . . . . .	25
4.2.1	Classes de javax.persistence . . . . .	26
4.2.2	Transaction . . . . .	26
4.2.3	Configuration . . . . .	26
4.2.4	persistence.xml . . . . .	26
4.2.5	Méthodes de javax.persistence . . . . .	27
4.2.6	JPQL . . . . .	28
4.3	Annotations . . . . .	30
4.4	Héritage . . . . .	30
4.5	Associations . . . . .	31
4.5.1	OneToOne . . . . .	31
4.5.2	OneToMany - ManyToOne . . . . .	32
4.5.3	ManyToMany . . . . .	32
4.5.4	Eager et Lazy . . . . .	33
4.6	DTO : Data Transfert Object . . . . .	33
<b>5</b>	<b>HTML : HyperText Markup Language</b>	<b>35</b>
5.1	Déclaration . . . . .	35
5.2	Mise en page . . . . .	35
5.3	Style CSS . . . . .	35
5.3.1	Attributs . . . . .	35
5.4	JavaScript [2] . . . . .	36
5.4.1	DOM : Document Object Model . . . . .	36
5.4.2	Fonctions . . . . .	37
5.4.3	. . . . .	37
5.5	JQuery [3] . . . . .	37

5.5.1	Déclaration . . . . .	37
5.5.2	Utilisation . . . . .	37
5.5.3	Correspondance . . . . .	38
5.5.4	Fonctions . . . . .	38
5.6	BootStrap . . . . .	38
<b>6</b>	<b>JSF : Java Server Faces</b>	<b>39</b>
6.1	Structures . . . . .	39
6.2	Configuration . . . . .	39
6.2.1	web.xml . . . . .	39
6.2.2	faces-config.xml . . . . .	40
6.2.3	ManagedBean . . . . .	40
6.2.4	Facelets . . . . .	41
6.2.5	Unified Expression Language . . . . .	41
6.3	Le cycle de vie . . . . .	41
6.3.1	Interfaces graphiques . . . . .	41
6.3.2	Options additionnelles . . . . .	41
6.4	Utilisation . . . . .	41
<b>7</b>	<b>HTTP : Hypertext Transfer Protocol</b>	<b>42</b>
7.1	Service REST . . . . .	42
7.2	Code erreur . . . . .	43
7.3	IP . . . . .	43
<b>8</b>	<b>ElasticSearch</b>	<b>44</b>
8.1	Introduction . . . . .	44
8.1.1	Qu'est qu'un moteur de recherche . . . . .	44
8.1.2	Token . . . . .	44
8.1.3	Base . . . . .	44
8.1.4	Historique . . . . .	45
8.1.5	Apache Lucene . . . . .	45
8.1.6	SGBDR vs Elasticsearch . . . . .	45
8.1.7	Principe . . . . .	46
8.2	Installation . . . . .	46
8.3	Elasticsearch-head . . . . .	46
8.4	Création/suppression . . . . .	46
8.5	Distribué/haute disponibilité . . . . .	47
8.5.1	Sharding . . . . .	47
8.5.2	Indexation . . . . .	47
8.5.3	Master . . . . .	47
8.6	Configuration . . . . .	47
8.7	Type de données . . . . .	47
8.8	Mappings/Settings . . . . .	48
8.9	Batch API : _bulk . . . . .	49
8.10	API de recherche : _search . . . . .	49

8.11	API _cat . . . . .	50
8.12	Routing . . . . .	50
8.13	_dynamic . . . . .	51
8.14	_source . . . . .	51
8.15	Query/Filter . . . . .	51
8.16	Full-Text . . . . .	52
8.17	GéoLocalisation . . . . .	52
8.18	Agrégations . . . . .	53
8.18.1	Agrégations imbriquées . . . . .	53
8.18.2	Metrics aggregations . . . . .	53
8.18.3	Bucket aggregations . . . . .	54
8.19	Java project . . . . .	55
<b>9</b>	<b>Kafka</b>	<b>56</b>
9.1	Intro . . . . .	56
9.1.1	Role . . . . .	56
9.1.2	Utilisation . . . . .	56
9.1.3	Stream . . . . .	57
9.1.4	Terminologie . . . . .	57
9.1.5	ZooKeeper . . . . .	57
9.1.6	TP Lab1 . . . . .	58
9.2	Architecture . . . . .	59
9.2.1	Consumer groups . . . . .	59
9.2.2	Stratégie . . . . .	59
9.2.3	Garanties . . . . .	59
9.3	Développer avec Kafka . . . . .	60
9.3.1	API Producer . . . . .	60
9.3.2	L'Ack d'un message . . . . .	61
9.3.3	API Consumer . . . . .	61
9.4	Zero Copy . . . . .	62
9.5	Serializer et Deserializer . . . . .	62
9.5.1	Librairie Jackson . . . . .	62
9.6	Confluent . . . . .	63
9.6.1	Kafka connect . . . . .	63
9.6.2	Kafka Streams . . . . .	63
9.7	Utilisation d'un connecteur Kafka Connect . . . . .	63
9.7.1	Étape 1 . . . . .	63
9.7.2	Étape 2 . . . . .	63
9.8	Kafka en python . . . . .	64
9.9	Kafka avec spark . . . . .	64
9.10	Autres . . . . .	64
<b>10</b>	<b>Python</b>	<b>65</b>
10.1	Introduction . . . . .	65

10.2 Environnement virtuel . . . . .	65
10.2.1 Matplotlib . . . . .	66
10.2.2 REST . . . . .	66
10.3 Programmation fonctionnelle . . . . .	66
10.3.1 lambda . . . . .	67
10.3.2 map . . . . .	67
10.3.3 filter . . . . .	67
10.3.4 reduce . . . . .	67
10.3.5 list comprehension . . . . .	67
10.4 Exception . . . . .	67
10.5 Classes . . . . .	68
10.6 Générateur . . . . .	68
10.7 Géolocalisation . . . . .	68
10.7.1 Obtenir coordonnées GPS . . . . .	68
10.7.2 Calcul de distance . . . . .	69
10.8 TP Velov . . . . .	69
10.9 TP Tweeter . . . . .	69
10.10 Git . . . . .	70
10.11 Crawler/Scraper . . . . .	71
10.12 Kafka . . . . .	72
10.13 Script Shell . . . . .	72
<b>11 Scala . . . . .</b>	<b>73</b>
11.1 Introduction . . . . .	73
11.2 Core concepts . . . . .	74
11.2.1 Recursion . . . . .	74
11.2.2 fonction d'ordre supérieur . . . . .	74
11.2.3 list . . . . .	75
11.2.4 match . . . . .	75
11.2.5 map/filter . . . . .	75
11.2.6 fold . . . . .	76
11.3 Scala concepts . . . . .	76
11.3.1 sealed . . . . .	76
11.3.2 Traits . . . . .	76
11.4 Theoretical . . . . .	76
11.4.1 Impératif - Procédural . . . . .	76
11.4.2 Lazy evaluation . . . . .	77
11.4.3 Pour et contre . . . . .	77
11.5 SBT : Simple Build Tool . . . . .	77
11.6 JSON . . . . .	78
<b>12 Hadoop : High-Availability Distributed Object-Oriented Platform . . . . .</b>	<b>79</b>
12.1 Intro . . . . .	79
12.1.1 Installation . . . . .	80

12.2	Yarn : Yet Another Resource Negotiator . . . . .	80
12.3	HDFS : Hadoop Distributed File System . . . . .	80
12.3.1	HDFS . . . . .	81
12.4	MapReduce . . . . .	81
12.5	Hive . . . . .	82
12.5.1	Lab : Hive CSV SerDe . . . . .	82
12.5.2	Lab : Hive JSON SerDe . . . . .	82
12.6	Pig . . . . .	82
12.6.1	Lab . . . . .	83
12.7	Sqoop . . . . .	83
12.7.1	Lab . . . . .	83
12.8	Oozie . . . . .	84
12.8.1	Lab . . . . .	84
12.9	HBase . . . . .	84
12.10	Solr/Lucene . . . . .	85
12.11	TP . . . . .	85
12.11.1	Oozie avec Pig et Hive . . . . .	85
12.11.2	Sqoop . . . . .	85
<b>13</b>	<b>Spark</b>	<b>87</b>
13.1	Introduction . . . . .	87
13.1.1	Installation . . . . .	87
13.2	Fonctions . . . . .	87
13.3	DAG : Directed Acyclic Graph . . . . .	88
13.3.1	Création d'un projet avec IntelliJ . . . . .	89
13.4	TP : Spark RDD . . . . .	89
13.5	ReduceByKey VS GroupByKey . . . . .	91
13.6	Cache . . . . .	91
13.7	Accumulator . . . . .	91
13.8	Broadcast . . . . .	91
13.9	Exception Scala . . . . .	92
13.10	TP 3 . . . . .	92
<b>14</b>	<b>MachineLearning</b>	<b>95</b>
14.1	Introduction . . . . .	95
14.1.1	Approche . . . . .	95
14.1.2	Problèmes fondamentaux . . . . .	95
14.1.3	Apprentissage supervisé . . . . .	95
14.1.4	Apprentissage non-supervisé . . . . .	95
14.1.5	Apprentissage renforcé . . . . .	96
14.1.6	Induction VS Dédution . . . . .	96
14.1.7	Bilan . . . . .	96
14.2	Scikit-learn . . . . .	96
14.2.1	Cloud . . . . .	96

14.2.2	datasets . . . . .	97
14.2.3	Réponses de l'apprentissage supervisé . . . . .	97
14.2.4	Pipelines standards . . . . .	97
14.2.5	Rasoir d'Occam . . . . .	97
14.3	Features . . . . .	97
14.3.1	Observations série-temporelles . . . . .	97
14.3.2	MFCCs : traitement du son . . . . .	97
14.3.3	Haar features : reconnaissance des images . . . . .	98
14.3.4	Bag of words : reconnaissance de texte . . . . .	98
14.4	Construction de modèles . . . . .	98
14.5	Metrics . . . . .	99
14.6	Learning Curves . . . . .	99
14.7	TP Fleurs . . . . .	99
14.8	Apprentissage non-supervisé . . . . .	99
14.9	TP Tweets . . . . .	100
14.9.1	Bag of words : CountVectorizer . . . . .	100
14.9.2	Tfidf : TfidfVectorizer . . . . .	101
14.9.3	Méthodologie . . . . .	101
14.9.4	Validation croisée . . . . .	101
<b>15</b>	<b>Visualisation</b>	<b>103</b>
15.1	Spark Notebook . . . . .	103
15.1.1	TP CO2 . . . . .	104
15.1.2	Diagramme . . . . .	104
15.2	Kibana . . . . .	104
<b>16</b>	<b>Base de données NoSQL</b>	<b>105</b>
16.1	Histoire . . . . .	105
16.1.1	Two forms of Big volumes . . . . .	106
16.1.2	Making sense at scale . . . . .	106
16.1.3	Crowdsourcing and Human computation . . . . .	106
16.2	NoSQL : Not only SQL . . . . .	106
16.2.1	Motivation . . . . .	107
16.2.2	Catégories . . . . .	107
16.2.3	ACID . . . . .	107
16.3	TP1 . . . . .	108
16.4	MongoDB . . . . .	108
16.4.1	Prise en main . . . . .	109
16.4.2	Insertion . . . . .	109
16.4.3	Sélecteurs . . . . .	110
16.4.4	Projections . . . . .	111
16.5	TP2 . . . . .	111
16.6	Agrégation . . . . .	112
16.7	Indexation . . . . .	112



16.8 TP2 . . . . .	112
16.8.1 Agrégation . . . . .	113
16.8.2 MapReduce . . . . .	113
16.8.3 Indexation . . . . .	114
16.9 Administration – API Java . . . . .	115
16.9.1 Administrer une base de données MongoDB . . . . .	115
16.10 Réplication . . . . .	117
16.11 CAP . . . . .	117
16.12 Verrou . . . . .	118
16.13 Évaluation . . . . .	118
<b>17 TP Spark</b>	<b>121</b>
17.0.1 Self-Contained Applications . . . . .	122
<b>18 Spark Streaming</b>	<b>124</b>
18.1 Introduction . . . . .	124
18.1.1 Event Sourcing . . . . .	124
18.2 Spark Streaming . . . . .	125
18.2.1 Principe . . . . .	125
18.2.2 Transformation . . . . .	125
18.2.3 Action . . . . .	125
18.3 TP . . . . .	126
18.3.1 Objet compagnon . . . . .	126
<b>19 Spark SQL</b>	<b>130</b>
19.1 DataFrames . . . . .	130
19.1.1 JSON . . . . .	130
19.1.2 Sauvegarde . . . . .	130
19.1.3 CSV . . . . .	131
19.2 Lab1 . . . . .	131
19.3 Lab2 . . . . .	132
19.4 Requêtes SQL . . . . .	133
19.5 DataSets . . . . .	133
19.5.1 Utilisation . . . . .	134
19.5.2 Correspondance . . . . .	134
19.5.3 Lab3 . . . . .	134
<b>20 Spark ElasticSearch</b>	<b>136</b>
20.1 Lab . . . . .	137
20.2 Kibana . . . . .	137
20.3 Quiz . . . . .	138
<b>21 D3JS</b>	<b>139</b>
<b>22 Projet</b>	<b>140</b>
22.1 Twitter Streaming . . . . .	140

22.2 Twitter Search . . . . .	140
22.3 Spark ML . . . . .	140
<b>Bibliographie</b>	<b>141</b>



# Chapitre 1

## Java

### 1.1 Eclipse

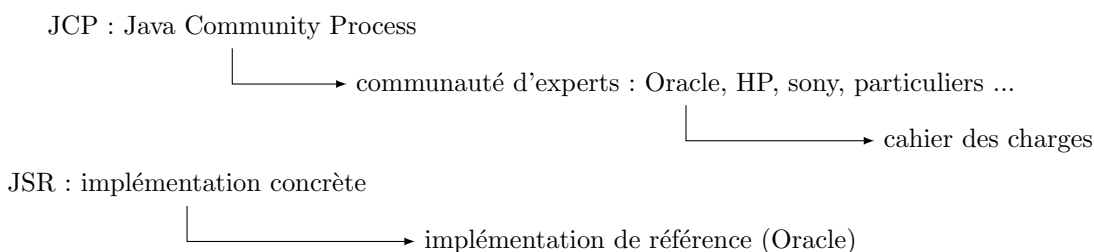
Il existe trois grand types d'applications informatiques.

- appli desktop : Swing (Java)
- appli web : JEE
- appli mobile : J2ME (Android)

Une application peut-être soit statique soit dynamique à l'aide d'une base de données. Il existe plusieurs version de Java.

1. Java SE (Standard Edition) : usage basique
2. Java EE (Enterprise Edition) : appli web, site internet

#### 1.1.1 Schéma



#### 1.1.2 Outils

- Project -> src -> new -> class
- Ctrl+espace pour lister les suggestions. Il s'agit de l'API reflection qui réalise une introspection sur tous les champs de Java.
- Ctrl+Shift+O pour gérer les imports.
- Ctrl+Shift+L pour lister les raccourcis.
- Alt+Shift+M pour extraire un morceau de code dans une nouvelle méthode.

#### 1.1.3 Mode debug

Pour passer en mode debug, il suffit de faire un marquer dans la marge de notre code. La compilation s'effectuera normalement jusqu'à arriver au marquer. La compilation passe alors en mode manuel étape par étape avec la touche F6. Il est ainsi possible de suivre la compilation et d'intercepter l'erreur.

## 1.2 Java

Java est une VM (Java Virtuel Machine) et n'utilise donc pas la mémoire directement comme en C. La JVM utilise un Garbage Collector qui gère l'utilisation de la mémoire. Le Ctrl+S provoque la compilation en `fichiers.class` à l'aide de l'outil *javac*. L'exécution est réalisée par l'outil *java*. Le langage Java est un langage orienté objet. Les constructeurs vide sont implicites.

Pour instancier une méthode ou une variable :

```
1 new methode();
```

Primitif ( défaut = 0 )	Objet ( défaut = null )
int	Integer
char	Character
long	Long
boolean	Boolean
double (64bit)	Double
short (16bit)	Short

### 1.2.1 Type

**public** est vue par tous les packages.

**private** doit avoir des getteurs et setteurs pour être appelé ou modifié.

**protected** sont seulement visibles dans le package courant.

**final** permet de fixer la valeur d'une variable. Une classe final ne peut plus être étendue.

**static** on peut appeler n'importe quelle méthode static en appelant la classe et non en créant un objet pour l'utiliser.

## 1.3 Héritage

L'héritage facilite la maintenance et l'évolution.

déclaration

```
1 public class Etudiant extends Scolaire {
2 }
```

Déclarer une classe abstract permet de ne pas pouvoir l'instancier. Elle n'a donc pas besoin de constructeur. Elle peut posséder des méthodes avec ou sans corps.

En Java, on ne peut hériter que d'une seule classe mais on peut implémenter plusieurs interfaces. Une interface est une classe abstraite. Cela permet donc de déclarer toutes les méthodes nécessaires sans les définir.

Le mot clé `super` permet d'appeler la classe mère. Le mot clé `this` permet d'appeler la classe courante.

`@inheritDoc` pour hériter de la doc mère.

### 1.3.1 Polymorphisme

On peut caster un objet de type A en type B seulement si la classe B hérite de A. Il est alors possible de récupérer les méthodes des classes mères.

exemple

```
1 Document doc = new CD();
2 (CD)doc.setNbPiste(12);
```

On instancie un document de type CD. Par défaut, nous avons uniquement accès aux attributs de la classe Document. `setNbPiste` est une méthode de la classe CD. Pour l'utiliser, il faut caster doc comme CD.

Il est préférable d'utiliser un polymorphisme plutôt que des switch.

## 1.4 Interface

Les interfaces commencent par I suivi du nom d'une classe.

déclaration

```
1 public class Etudiant extends Scolaire implements IScolaire {
2     }
```

L'utilisation est de réunir les différentes méthodes dans une interface mère sous un même nom. L'implémentation diffèrera ou non dans les classes filles différentes. A partir de Java 8, l'implémentation peut se faire dans une interface.

Toute classe implémentant une interface doit implémenter toutes les méthodes de cette interface. Dans le cas d'une classe abstraite implémentant une interface, nous avons le choix d'implémenter aucune, une, plusieurs ou toutes les méthodes de l'interface. Cette classe abstraite ne peut cependant pas être instanciée directement. Il faut pour cela avoir une classe qui étend cette classe abstraite. Les éventuelles méthodes non implémentées dans la classe abstraite devront être implémentées dans cette classe fille.

## 1.5 Enumeration

déclaration

```
1 public enum Niveaux {
2     primaire, secondaire, superieur;
3 }
```

exemple

```
1 private Niveaux niveau;
2 niveau = Niveaux.secondeire;
```

## 1.6 Généricité

déclaration

```
1 public class Generique<T> {
2     protected int numero;
3     private T generic;
4
5     public Generique() { //constructeur
6         super();
7     }
8     public Generique(int numero, T generic) { //constructeur avec parametres
9         super();
10        this.numero = numero;
11        this.generic = generic;
12    }
13
14    public T getGeneric() { //getteur
15        return generic;
16    }
17    public void setGeneric(T generic) { //setteur
18        this.generic = generic;
19    }
20 }
```

La variable generic de type T prend le type de l'implémentation que l'on crée.

## 1.7 Collection

tableaux = taille fixe

déclaration

```
1 double[] nomTab = new double[4];
```

liste = taille dynamique

### 1.7.1 List

déclaration

```
1 List<type> nomListe = new ArrayList<>();
```

Le type peut être un entier, une chaîne, une énumération ou toute autre classe préalablement créée.

Utilisation

```
1 nomListe.add(element) //ajoute un element du bon type
2 nomListe.addAll(collection) //ajoute une collection
3 nomListe.contains(element) //test de recherche
```

### 1.7.2 Set

Création d'une liste qui n'accepte pas les doublons.

déclaration

```
1 Set<type> nomListe = new HashSet();
```

### 1.7.3 Map

Déclaration

```
1 Map<key, type> nomMap = new HashMap<>();
```

Utilisation

```
1 nomMap.put(key, variable) //ajoute un element
2 temp = nomMap.get(key) //retourne l'element reference par key
```

## 1.8 Opération ternaire

Syntaxe : (condition) ? x : y ;

avant

```
1 if (count == 1) {
2     return true;
3 }else {
4     return false;
5 }
```

après

```
1 return (count == 1)? true:false;
```

avant

```

1  int x = 10;
2  int y = 20;
3  if (x < y) {
4      max = y;
5  }else {
6      max = x;
7  }

```

après

```

1  int x =10;
2  int y =20;
3  int max = (x<y)?y:x;

```

## 1.9 IO Stream

déclaration

```

1  String str = '';
2  try {
3      InputStreamReader isr = new InputStreamReader(System.in);
4      BufferedReader br = new BufferedReader(isr);
5      str = br.readLine(); //lecture de la saisie
6  }catch (Exception e) {
7      e.printStackTrace();
8  }

```

**Stream** envoie octet par octet les données. Il faut donc utiliser un buffer pour collecter les données avant d'envoyer le paquet final.

### 1.10 Exceptions

La gestion des exceptions peut se faire de deux façons :

- **try/catch** afin d'isoler l'exception et de la traiter sans planter le programme
- mention **throws** Exception pour passer l'exception à la méthode appelante

Déclaration

```

1  public void statistiques() throws Exception {
2  }

```



## Chapitre 2

# SGBDR : Base de données Relationnelle

Il existe plusieurs bases de données : PostgreSQL, MySQL, SQL Server, Oracle, Access (cf : w3schools). L

### 2.1 Formes normales

1<sup>ère</sup> forme normale : chaque information est atomique. 2 et 3<sup>ème</sup> forme normale : minimum de redondance d'information.

### 2.2 Requête SQL

#### 2.2.1 Les livraisons

Soit la base relationnelle de données livraison de schéma :

- Usine (id, nom, ville)
- Produit (id, nom, couleur, poids)
- Fournisseur (id, nom, statut, ville)
- Livraison (produit\_id, usine\_id, fournisseur\_id, quantité)

Une usine est décrite par son numéro, son nom et la ville. Un produit est décrit par son numéro, son nom, sa couleur et son poids. Un fournisseur est décrit par son numéro, son nom, son statut (sous-traitant, client...) et la ville où il est domicilié. Une livraison est décrite par le produit de numéro **produit\_id** délivré à l'usine de numéro **usine\_id** par le fournisseur de numéro **fournisseur\_id** dans une quantité donnée.

1. Ajouter un nouveau fournisseur avec les attributs de votre choix

```
1 INSERT INTO fournisseur (nom, ville) VALUES (?,?)
```

2. Supprimer tous les produits de couleur noire et de numéros compris entre 100 et 1999

```
1 DELETE FROM produit WHERE couleur = 'noir' AND np<=1999 AND np>=100
```

3. Changer la ville du fournisseur 3 par Toulouse

```
1 UPDATE fournisseur SET ville = 'toulouse' WHERE id = 3
```

4. Donnez le numéro, le nom, la ville de toutes les usines

```
1 SELECT * FROM usine
```

5. Donnez le numéro, le nom, la ville de toutes les usines de Paris

```
1 SELECT * FROM usine WHERE ville = 'Paris'
```



6. Donnez les numéros des fournisseurs qui approvisionnent l'usine de numéro 2 en produit de numéro 100

```
1 SELECT
```

7. Donnez les noms et les couleurs des produits livrés par le fournisseur de numéro 2

```
1 SELECT nom, couleur FROM produit WHERE id_produit IN
2       (SELECT produit_id FROM puf WHERE fournisseur_id = 2)
```

8. Donnez les numéros des fournisseurs qui approvisionnent l'usine de numéro 2 en un produit rouge

```
1 SELECT fournisseur_id FROM livraison WHERE usine_id = 2 AND produit_id IN
2       (SELECT id_produit FROM produit WHERE couleur = 'rouge')
```

9. Donnez les noms des fournisseurs qui approvisionnent une usine de Paris ou de Créteil en produit rouge

```
1 SELECT nom FROM fournisseur WHERE id_fournisseur IN
2       (SELECT fournisseur_id FROM livraison WHERE produit_id IN
3        (SELECT id_produit FROM produit WHERE couleur = 'rouge') AND
4        usine_id IN (SELECT id_usine FROM usine WHERE ville = 'Paris' OR 'Creteil'))
```

10. Donnez les numéros des produits livrés à une usine par un fournisseur de la même ville

```
1 SELECT numP FROM p, u, f, puf WHERE puf.numP = p.numP AND
2       puf.numF = f.numF AND puf.numU = u.numU AND u.ville = f.ville
3
4 ou
5
6 SELECT id_produit FROM produit WHERE id_produit IN
7       (SELECT produit_id FROM livraison WHERE fournisseur_id IN
8        (SELECT id_fournisseur FROM fournisseur f) AND usine_id IN
9        (SELECT id_usine FROM usine u) AND u.ville = f.ville)
```

11. Donnez les numéros des produits livrés à une usine de Paris par un fournisseur de Paris.

```
1 SELECT numP FROM p, u, f, puf WHERE puf.numP = p.numP AND
2       puf.numF = f.numF AND puf.numU = u.numU AND
3       u.ville = 'Paris' AND f.ville = 'Paris'
```

12. Donnez les numéros des usines qui ont au moins un fournisseur qui n'est pas de la même ville

```
1 SELECT numP FROM p, u, f, puf WHERE puf.numP = p.numP AND
2       puf.numF = f.numF AND puf.numU = u.numU AND
3       u.ville <> f.ville
```

13. Donnez les numéros des fournisseurs qui approvisionnent à la fois des usines de numéros 2 et 3

```
1 SELECT fournisseur_id FROM livraison WHERE fournisseur_id IN
2       (SELECT fournisseur_id FROM livraison WHERE usine_id = 2) AND fournisseur_id IN
3       (SELECT fournisseur_id FROM livraison WHERE usine_id = 7);
```

14. Donnez les numéros des usines qui utilisent au moins un produit disponible chez le fournisseur de numéro 3 (c'est-à-dire un produit que le fournisseur livre mais pas nécessairement à cette usine)

```
1 SELECT usine_id FROM livraison WHERE produit_id IN
2       (SELECT produit_id FROM livraison WHERE fournisseur_id = 3);
```

15. Donnez le numéro du produit le plus léger (les numéros si plusieurs produits ont ce même poids)

```
1 SELECT id_produit FROM produit WHERE poids IN (SELECT MIN(poids) FROM produit);
```

16. Donnez le numéro des usines qui ne reçoivent aucun produit rouge d'un fournisseur parisien

```
1 SELECT usine_id FROM livraison WHERE produit_id NOT IN
2       (SELECT id_produit FROM produit WHERE couleur = 'rouge' AND id_produit IN
3        (SELECT produit_id FROM livraison WHERE fournisseur_id IN
4        (SELECT id_fournisseur FROM fournisseur WHERE ville = 'paris'))));
```

17. Donnez les numéros des fournisseurs qui fournissent au moins un produit fourni par au moins un fournisseur qui fournit au moins un produit rouge

```
1 SELECT fournisseur_id FROM livraison WHERE produit_id IN
2       (SELECT produit_id FROM livraison WHERE fournisseur_id IN
3        (SELECT fournisseur_id FROM livraison WHERE produit_id IN
4        (SELECT id_produit FROM produit WHERE couleur = 'rouge'))));
```

Voir plus de fonctions au § 4.2.6 et 4.2.6.

### 2.2.2 La société Gavasoft

Soit les relations suivantes de la société Gavasoft.

- Employe (id, nom, fonction, numS, embauche, salaire, commission, departement\_id)
- Departement (id, nom, lieu)

Exemple

id_departement	nom	lieu
1	Droit	Paris
2	Commerce	Boston

nom	fonction	numS	embauche	salaire	commission	dept_id
Gava	Président	NULL	10/10/1979	10000	NULL	NULL
Guimezanes	Doyen	1	01/10/2006	6000	NULL	1
Tot	Stagiaire	1	01/10/2006	0	NULL	1
Al-Capone	Commercial	2	01/10/2006	4000	100	2

1. Donnez la liste des employés ayant une commission (non NULL) classé par commission décroissante

```
1 SELECT nom FROM employe WHERE commission > 0 ORDER BY commission DESC;
```

2. Donnez les noms des personnes embauchées depuis le 01-09-2006

```
1 SELECT nom FROM employe WHERE embauche > 2006/09/01 ORDER BY nom ASC;
```

3. Donnez la liste des employés travaillant à Créteil

```
1 SELECT * FROM employe WHERE departement_id IN
2 (SELECT id_departement FROM departement WHERE lieu='Paris');
```

4. Donnez la liste des subordonnés de "Guimezanes"

```
1 SELECT * FROM employe WHERE numS IN
2 (SELECT numS FROM employe WHERE nom = 'Guimezanes') AND nom <> 'Guimezanes';
```

5. Donnez la moyenne des salaires

```
1 SELECT AVG(salaire) FROM employe;
```

6. Donnez le nombre de commissions non NULL

```
1 SELECT COUNT(commission) FROM employe;
```

7. Donnez la liste des employés gagnant plus que la moyenne des salaires de l'entreprise

```
1 SELECT * FROM employe WHERE salaire > (SELECT AVG(salaire) FROM employe);
```

La requête DELETE est de moins en moins utilisée. Après un certain temps, les données utilisateurs peuvent être vendues à des sociétés commerciales. Une colonne *actif* est ajouté pour discriminer par **true** ou **false** afin de ne pas afficher les utilisateurs "supprimés" tout en les gardant dans la base de données. Une méthode **delete** est donc remplacée par un **update** changeant la valeur de la colonne *actif*.

### 2.2.3 TP Médicaments

Compter les doublons

```
1 SELECT cis, codesub FROM bdpm_compo GROUP BY cis, codesub HAVING count(*) > 1;
```

```
1 EXPLAIN SELECT cip7, nomsb FROM bdpm_ciscip AS cip, bdpm_compo AS co
2 WHERE cip.cis = co.cis AND element LIKE "comprim%";
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	co	NULL	ALL	NULL	NULL	NULL	NULL	25536	11.11	Using where
1	SIMPLE	ci	NULL	ALL	NULL	NULL	NULL	NULL	19239	10.00	Using where; Using join buffer (Block Nested Loop)

2 rows in set, 1 warning (0.00 sec)

## Création d'un index

```
1 CREATE INDEX element_idx ON bdpm_compo(element);
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	ci	NULL	ALL	NULL	NULL	NULL	NULL	19239	11.11	NULL
1	SIMPLE	co	NULL	ALL	element_idx	NULL	NULL	NULL	25536	10.00	Using where; Using join buffer (Block Nested Loop)

2 rows in set, 1 warning (0.00 sec)

Afin d'optimiser la requête, une méthode serait de ne pas avoir de jointure. Pour ce faire, il faut ajouter les colonnes utilisées d'une table vers l'autre table.

## Création d'une nouvelle colonne

```
1 ALTER TABLE bdpm_compo ADD column cip7 int(11);
```

Réaliser un UPDATE sur des milliers de lignes prend trop de temps. Pour optimiser la mise à jour, on crée des index sur les colonnes utilisées.

## Update

```
1 CREATE INDEX cis_ciscipidx ON bdpm_ciscip(cis);
2 CREATE INDEX cis_compoidx ON bdpm_compo(cis);
3 UPDATE bdpm_compo AS co, bdpm_ciscip AS ci SET co.cip7 = ci.cip7 WHERE co.cis = ci.cis;
```

## 2.3 MySQL Workbench

### 2.3.1 Diagramme

DataBase -> Reverse Engineer

## 2.4 MariaDB/MySQL

Créer un utilisateur ayant les droits minimum juste pour de la sauvegarde

```
1 CREATE USER 'botbackup'@'localhost' IDENTIFIED BY 'password';
2 GRANT SELECT, SHOW VIEW, RELOAD, REPLICATION CLIENT, EVENT, TRIGGER ON *.* TO 'botbackup'@'localhost';
3 GRANT LOCK TABLES ON *.* TO 'botbackup'@'localhost';
```

### 2.4.1 Utilisation des GRANT

La clause SQL GRANT vous donne la possibilité d'attribuer/révoquer soit des **System Privileges**, soit des **Objects Privileges**.

## Syntaxe de base pour attribuer un privilège système

```
1 GRANT privilege(s)_systeme_attribue(s) TO utilisateur(s)_vise(s)
2 (WITH ADMIN OPTION)
```

L'option WITH ADMIN OPTION n'est pas obligatoire. Si elle est attribuée à un utilisateur, celui-ci pourra attribuer/révoquer ces propres privilèges systèmes sur un nouvel utilisateur.

Voici une liste (non-exhaustive) des différents privilèges systèmes existants :

- CREATE SESSION : Permet à un utilisateur de pouvoir créer une session avec la base de données.
- CREATE TABLE : Permet à l'utilisateur de créer des tables.
- ALTER DATABASE : Permet à l'utilisateur de modifier la structure d'une base de données.
- CREATE VIEW : Permet à l'utilisateur de créer des vues.
- ALL PRIVILEGES : Donne à l'utilisateur tous les privilèges systèmes existants (à utiliser avec modération!).

Autoriser lolokai à se connecter à des bases de données et de créer des vues

```
1 GRANT create session, create view TO lolokai
```

Syntaxe pour révoquer le(s) privilège(s) système d'un utilisateur

```
1 REVOKE privilege(s)_systeme_a_supprime(s) FROM utilisateur(s)_vise(s)
```

Attention, pour pouvoir le faire, il faut que vous possédiez l'option `WITH ADMIN OPTION` et que vous vouliez supprimer un privilège système que vous avez vous même attribué.

Comme nous l'avons vu précédemment, contrairement aux **System Privileges**, les **Objects Privileges** permettent de donner des droits aux utilisateurs pour modifier le contenu (les données) de nos bases de données.

Syntaxe de base pour attribuer un privilège objet

```
1 GRANT privilege(s)_objet_attribue(s) ON table_cible TO utilisateur(s)_cible(s)
2 (WITH GRANT OPTION)
```

Tout comme pour l'option `WITH ADMIN OPTION`, l'option `WITH GRANT OPTION` est optionnelle et permet à un utilisateur d'attribuer/révoquer un privilège à un autre utilisateur.

Voici quelques privilèges objet que vous pouvez attribuer (liste non-exhaustive) :

- `INSERT` : Permet d'insérer une nouvelle ligne dans une table.
- `SELECT` : Permet d'interroger une table.
- `DELETE` : Permet de supprimer des lignes d'une table.
- `UPDATE` : Permet de mettre à jour les lignes d'une table.
- `ALTER` : Permet de modifier la structure d'une table.

Autoriser lolokai à interroger et supprimer des lignes de la table Alpha

```
1 GRANT select, delete ON alpha FROM lolokai
```

Révoquer le(s) privilège(s) objet d'un utilisateur

```
1 REVOKE privilege(s)_objet_a_supprime(s) ON table_cible FROM utilisateur(s)_cible(s)
```

Attention, pour pouvoir le faire, il faut que vous possédiez l'option `WITH GRANT OPTION` et que vous vouliez supprimer un privilège objet que vous avez vous même attribué. Si vous supprimez le privilège de l'un de vos utilisateurs, tous les utilisateurs-enfants de celui-ci perdront aussi ce droit.

Pour exemple, imaginons que l'utilisateur lolokai possède l'option `WITH GRANT OPTION` et qu'il donne un privilège objet à l'utilisateur site. Ce même privilège est transmis de l'utilisateur site à l'utilisateur blog. Si dans un moment de folie extrême, l'utilisateur lolokai révoque le privilège à l'utilisateur site, l'utilisateur blog perdra aussi ce droit.

Ce phénomène est appelé « suppression en cascade ».

## 2.4.2 Importer/Exporter

Exporter une base de données

```
1 mysqldump -u root --password=inti db_name > dump_db_name.sql
```

Importer une base de données

```
1 mysql -u root --password=inti db_name < dump_db_name.sql
```

Une base de données doit exister au préalable.



## Chapitre 3

# JDBC : Java Data Base Connectivity

C'est une API = ensemble de classes et d'interfaces. Elle permet de se connecter à une base de données et d'effectuer des opérations (Create Read Update Delete).



### 3.1 Classes et interfaces

#### 3.1.1 java.sql.DriverManager

Download mysql connector j (5.1.41)

1. prise en charge du chargement du pilote
2. création d'une connexion à la BD
3. méthodes : `Connection getConnection()`

déclaration

```
1 static Connection getConnection() throws SQLException {  
2     final String url = "jdbc:mysql://127.0.0.1:3306/gestion_bib";  
3     Connection conn = DriverManager.getConnection(url, "root", "alfred");  
4     return conn;  
5 }
```

#### 3.1.2 java.sql.Connection

1. représente une connexion à la BD
2. méthodes :
  - `close()` : fermer la connexion
  - `Statement createStatement()` : définir une requête vers la BD associée à la connexion
  - `PreparedStatement createPreparedStatement()` : requête précompilée (plus rapide)

#### 3.1.3 java.sql.Statement

1. transmission des instructions à la BD ( `SELECT`, `INSERT`, ...)
2. méthodes :
  - `boolean execute()` : exécuter une requête lorsque le contenu est inconnu
  - `int executeUpdate()` : insertion, suppression ou mise à jour. On retourne un entier correspond au nombre lignes modifiées.
  - `executeQuery()` : récupération des données

### 3.1.4 java.sql.ResultSet

1. ensemble de résultats récupérés de la BD
2. méthodes
  - `boolean next()` : pointeur pour la lecture du `ResultSet`
  - `getXXX()` : récupérer les données du `ResultSet`. Le XXX désigne le type de données à récupérer.

exemple

```

1  conn = newConnection();
2
3  String query = "SELECT * FROM exemplaire;";
4  PreparedStatement state = conn.prepareStatement(query); //prepare la requete
5  ResultSet res = state.executeQuery(query); //execute et recupere les resultats
6
7  while (res.next()) {
8      System.out.println(res.getString("reference") + res.getString("titre")
9                          + res.getString("etat") + res.getString("date_retour")
10                         + res.getString("date_fin_reservation"));
11  }
```

- `previous()` : déplace le pointeur sur la rangée précédente
- `first(), last()` ...

### 3.1.5 Vue avec eclipse

Vue de la BD intégrée avec Windows –> Show view –> Data source explorer

### 3.1.6 Configuration

clic droit –> new :

1. choisir le type MySQL
2. définir le driver et le chemin du jar
3. éditer les paramètres de connexion : nom de la BD et le mdp
4. test de la connexion

### 3.1.7 Scrapbook

1. choisir le profil de connexion
2. entrer les requêtes SQL

## 3.2 Séparation des couches

1. Main
2. Entités
3. Services
4. DAO

### 3.2.1 DAO

On utilise une interface DAO et une classe DaoImpl qui implémente notre interface. La définition des méthodes seront réalisées dans la classe DaoImpl.

1. Créer une connexion
  - (a) chargement du pilote

exemple

```

1  public void ajouterUtilisateur(Utilisateur pUser) {
2
3      Connection connexion = null;
4      PreparedStatement requeteJdbc = null;
5
6      try {
7          //1: chargement du pilote jdbc de mysql
8          Class.forName("com.mysql.jdbc.Driver");
9
10         //2: recuperation d'une connexion
11         connexion = DriverManager.getConnection(url, login, password);
12
13         //3: definition d'une requete
14         String addUserReq = "INSERT INTO utilisateur (nom, mail, password)"+
15                             "VALUES (?, ?, ?) ";
16
17         //4: creation de la requete
18         requeteJdbc = connexion.prepareStatement(addUserReq);
19
20         //5: passage des parametres
21         requeteJdbc.setString(1, pUser.getNom());
22         requeteJdbc.setString(2, pUser.getMail());
23         requeteJdbc.setString(3, pUser.getPassword());
24
25         //6: execution de la requete et recuperation du resultat
26         int resultat = requeteJdbc.executeUpdate();
27
28         //7: test du resultat
29         if (resultat == 1) {
30             System.out.println("Utilisateur ajoute");
31         } else {
32             System.out.println("erreur");
33         }
34
35         //8: fermeture de la requete et de la connexion
36         requeteJdbc.close();
37         connexion.close();
38
39     } catch (ClassNotFoundException e) {
40         //exception due au forName()
41         e.printStackTrace();
42     } catch (SQLException e) {
43         //exception due a la connexion
44         e.printStackTrace();
45     } finally {
46         try {
47             if (requeteJdbc != null)
48                 if (connexion != null) connexion.close();
49             } catch (SQLException e) {
50                 e.printStackTrace();
51             }
52         }
53     }

```



## Chapitre 4

# JPA : Java Persistence API

Il s'utilise à travers des ORM (Object Relationnel Mapping).

- EclipseLink
- Hibernate (JBoss)
- IBatis
- topLink

cf. <http://igm.univ-mlv.fr/~dr/XPOSE2007/acollign ORM-JPA/index.html#context>.

L'utilité est de ne plus interagir avec la base de données, on délègue au JPA les requêtes.

### 4.1 Hibernate

C'est un framework développé par JBoss (maintenant une division de Red Hat). L'utilité est d'automatiser la création des tables et des ses colonnes dans une base de données. Pour ce faire, Hibernate se base sur deux fichiers `cfg.xml` et `hbn.xml`. La connexion entre Hibernate et la base de données s'établit avec JDBC.

#### 4.1.1 hibernate.cfg.xml

Ce fichier contient les paramètres de connexion avec notre base de données.

```
1 <!DOCTYPE hibernate-configuration PUBLIC
2 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
3 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
4
5 <hibernate-configuration>
6   <session-factory>
7     <!-- hibernate dialect -->
8     <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
9     <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
10    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/db_test_hibernate</property>
11    <property name="hibernate.connection.username">root</property>
12    <property name="hibernate.connection.password">alfred</property>
13    <property name="transaction.factory_class">org.hibernate.transaction.JDBCTransactionFactory</property>
14    <!-- Automatic schema creation (begin) === -->
15    <property name="hibernate.hbm2ddl.auto">create</property>
16    <!-- JDBC connection pool (use the built-in) -->
17    <property name="connection.pool_size">1</property>
18    <!-- SQL dialect -->
19    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
20    <!-- Disable the second-level cache -->
21    <property name="cache.provider_class">org.hibernate.cache.internal.NoCacheProvider</property>
22    <!-- Echo all executed SQL to stdout -->
23    <property name="show_sql">true</property>
24    <mapping resource="employee.hbm.xml" />
25  </session-factory>
26 </hibernate-configuration>
```



### 4.1.2 hbm.xml

Ces fichiers contiennent le mapping de nos classes. Il faut donc créer un fichier `hbm.xml` par classe créée. Les clés primaires se déclare avec la propriété `class="native"`.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <!DOCTYPE hibernate-mapping PUBLIC
3   "-//Hibernate/Hibernate Mapping DTD//EN"
4   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
5
6 <hibernate-mapping>
7   <class name="fr.inti.entities.Employe" table="employe">
8     <meta attribute="class-description">
9       This class contains the employee detail.
10    </meta>
11    <id name="id" type="long" column="id">
12      <generator class="native" />
13    </id>
14    <property name="nom" column="nom" type="string" />
15    <property name="prenom" column="prenom" type="string" />
16    <property name="salaire" column="salaire" type="int" />
17  </class>
18 </hibernate-mapping>

```

### 4.1.3 Annotations

Le mapping `resource="employee.hbm.xml"` est utilisé pour mapper les classes créées en s'appuyant sur le fichier `hbm.xml`. Le mapping peut également se faire avec les annotations (cf. 4.3). Il faut donc pour cela déclarer `mapping class="nom de la classe"`. La déclaration de la base de données à considérer s'établit dans les différentes classes avec l'annotation `@Table`.

```

1 @Entity
2 @Table(name="user", catalog="db_gestion_magasin")
3 public class User {
4 }

```

### 4.1.4 Méthodes de hibernate

- save()
- update()
- delete()
- get()

#### Déclaration

```

1 try {
2     factory = new Configuration().configure().buildSessionFactory();
3 } catch (Throwable ex) {
4     System.err.println("Failed to create sessionFactory object." + ex);
5     throw new ExceptionInInitializerError(ex);
6 }
7 Session session = factory.openSession();
8 Transaction tx = null;
9 try {
10     tx = session.beginTransaction();
11     session.save(objet);
12     tx.commit();
13 } catch (HibernateException e) {
14     if (tx != null)
15         tx.rollback();
16     e.printStackTrace();
17 } finally {
18     session.close();
19 }

```

### 4.1.5 HQL

La syntaxe est similaire au SQL cf. 2.2 et JPQL cf. 4.2.6.

#### Déclaration

Après la déclaration de la connexion et de la transaction.

```
1 String querySql = "SELECT * FROM produit";
2 SQLQuery st = session.createSQLQuery(querySql);
3 liste = (List<Produit>) st.addEntity(Produit.class).list(); // recupere une liste
```

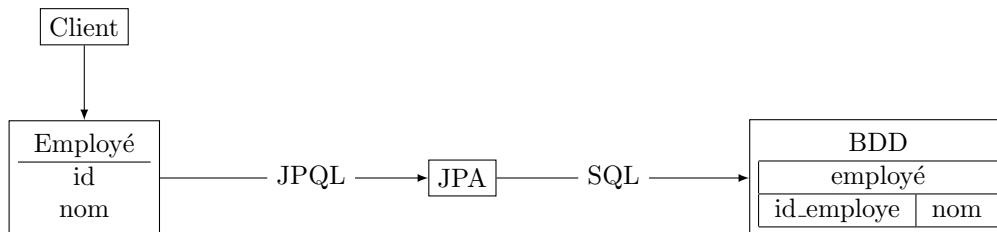
#### Paramètres

Le passage de paramètres dans une requête s'effectue avec le symbole : suivi du label du paramètre. Après création de la requête, il faut setter en utilisant `setParameter()` avec le label déclaré.

#### JDBC vs Hibernate

Connection ----- SessionFactory  
Statement ----- Session  
PreparedStatement ----- SQLQuery

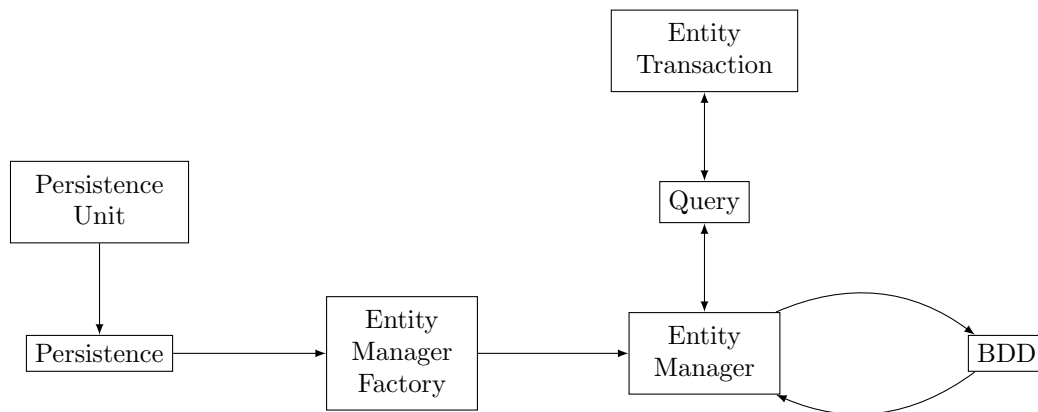
## 4.2 EclipseLink



1. Définir le Mapping JPA : class Java -> entité (i.e. table dans la BD )
2. Configuration :
  - (a) Persistence Unit
    - i. infos de connexion
    - ii. déclaration des entités
    - iii. propriétés JPA
      - génération auto des tables dans la BD

JPA est géré par le package `javax.persistence`.

### 4.2.1 Classes de javax.persistence



### 4.2.2 Transaction

Une transaction permet de réunir plusieurs requêtes. L'exécution des requêtes est soumise à la possibilité d'exécution de chaque requête. Si une requête ne passe pas, la transaction est annulée et aucune des requêtes n'est persistée. On utilise pour cela les méthodes suivantes :

1. `commit()` : confirmation de chaque requête
2. `rollback()` : annulation de la requête si une exception est levée.

exemple

```

1 try {
2     transaction
3     commit
4 } catch (Exception e) {
5     rollback
6 }
  
```

### 4.2.3 Configuration

project properties -> Project Facets -> JPA -> further configuration

1. choix de la platform : EclipseLink
2. JPA implementation :
  - (a) Type : User library pour laisser eclipse prendre ce qu'il faut.
  - (b) Download libraries
  - (c) choisir notre connexion : MySQL
  - (d) Valider

-> generation des librairies + *persistence.xml* dans *src*.

### 4.2.4 persistence.xml

Le XML (eXtended Markup Language) est un langage de balises. Les balises sont créées par un consortium W3C. Son utilisation principale est la transmission de données contrairement au HTML qui est orienté structure d'une page web.

exemple

```

1 <users>
2     <user age="20">
3         <id>1</id>
4         <nom>Sam</nom>
5     </user>
6 </users>
  
```

*id* et *nom* sont les attributs de la balise *user*. Ces derniers peuvent également être déclarés directement dans la balise ouvrante (i.e. *age*).

#### implémentation du JPA

```

1 <persistence-unit name="01-introJpa">
2   <!-- provider (implementation concrete de JPA : EclipseLink -->
3   <provider>org.eclipse.persistence.jpa.PersistenceProvider.class</provider>
4 </persistence-unit>

```

#### déclaration de la connexion

```

1 <!-- declaration des propriete JPA -->
2 <properties>
3   <!-- declaration des infos pour la connexion a la BDD -->
4   <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/db_test_jpa"/>
5   <property name="javax.persistence.jdbc.user" value="root"/>
6   <property name="javax.persistence.jdbc.password" value="alfred"/>
7   <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
8   <!-- mode de generation des tables -->
9   <property name="javax.persistence.schema-generation.database.action" value="create"/>
10  <!-- affichage dans la console des requetes SQL executes -->
11  <property name="eclipselink.logging.level" value="FINE"/>
12 </properties>

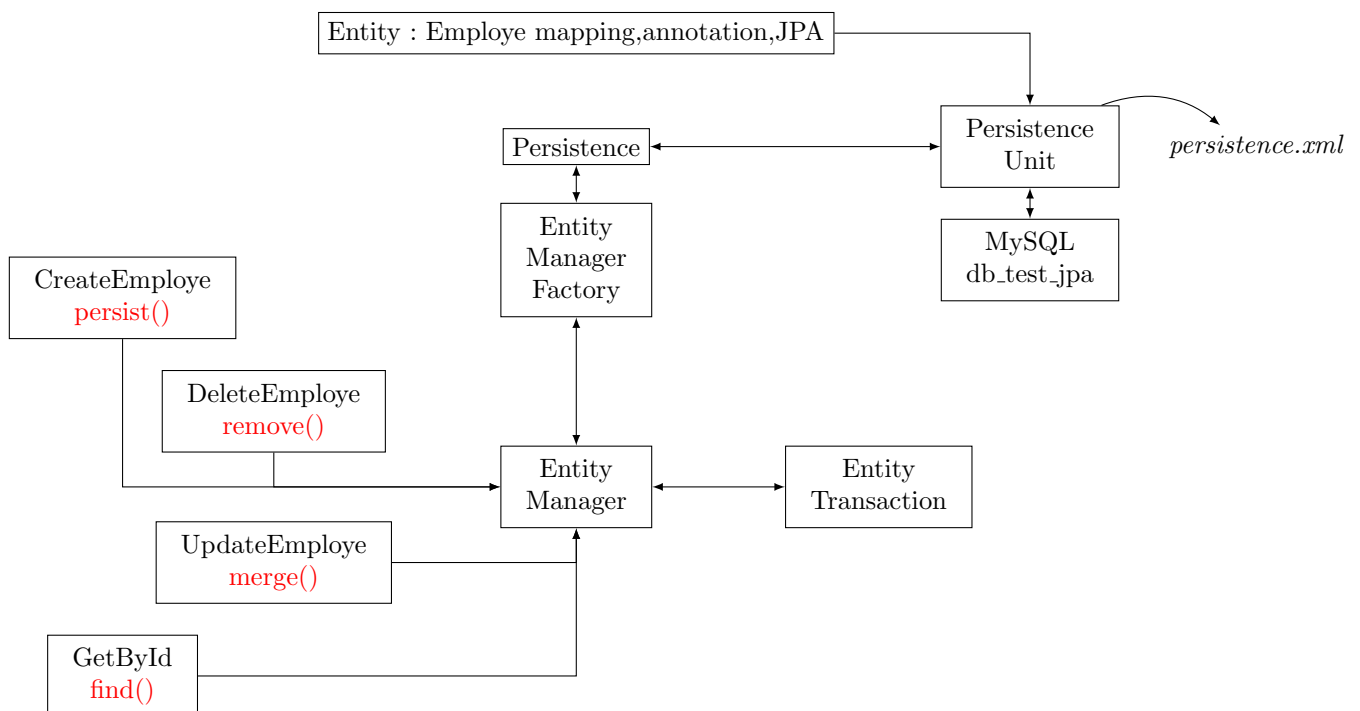
```

### Génération des tables

Dans notre *persistence.xml*, plusieurs mode de génération des tables peuvent être définies.

1. **none** : ne fais rien
2. **create** : créer la table et la remplit
3. **drop-and-create** : efface tout et recréer
4. **drop** : efface tout

### 4.2.5 Méthodes de javax.persistence



### 4.2.6 JPQL

Le JPA a son langage de requête, le JPQL : Java Persistence Query Language. La syntaxe est basé sur SQL. Toutefois, les requêtes se basent sur les noms des classes et leurs attributs.

- récupération des données : clause **SELECT**
- mise à jour des données : clause **UPDATE**
- suppression des données : clause **DELETE**

#### Syntaxe

cf. 2.2

##### 1. Select

```
1 SELECT ... FROM ...
2     WHERE ...
3     GROUP BY ... HAVING ...
4     ORDER BY ... ASC | DESC
```

##### 2. Update

```
1 UPDATE ... SET ... WHERE ...
```

##### 3. Delete

```
1 DELETE FROM ..(entite)... WHERE ...
```

#### Utilisation

Les requêtes sont écrites avec les noms des entités (i.e. classes java) au lieu des noms de la table SQL. De ce fait, la sélection d'une entité récupère toute la ligne. Pour récupérer un ou plusieurs attributs, on les instancie comme pour une classe java (i.e. entite.nom). La définition d'une requête se fait par la méthode `createQuery()`. La récupération du résultat peut se faire avec `getSingleResult()` ou `getResultList()`.

#### Fonctions

##### En-tête

```
1 //1.
2 EntityManager em =Persistence.createEntityManagerFactory("01-introJpa").createEntityManager();
```

##### 1. Upper / Lower

```
1 System.out.println("----- Fonction UPPER() | LOWER() -----");
2 // requete
3 String upperReq = "SELECT UPPER(e.nom) FROM employe e";
4 // execution + recup des noms des employes
5 List<String> listNomEmploye = em.createQuery(upperReq).getResultList();
6 // affichage
7 for (String val :listNomEmploye) {
8     System.out.println(" Nom \t : "+val);
9 }
```

##### 2. Count

```
1 System.out.println("----- Fonction COUNT() -----");
2 // requete
3 String countReq = "SELECT COUNT(e.id) FROM employe e";
4 // execution + recup
5 long verif = (long)em.createQuery(countReq).getSingleResult();
6 // affichage
7 System.out.println(" Nombre d'employe : "+verif);
```

## 3. Max

```

1 System.out.println("----- Fonction MAX() -----");
2 // requete
3 String maxReq = "SELECT MAX(e.salaire) FROM employe e";
4 // execution + recup
5 double salaireMax = (double)em.createQuery(maxReq).getSingleResult();
6 // affichage
7 System.out.println(" Salaire max : "+salaireMax);

```

## 4. Average

```

1 System.out.println("----- Fonction AVG() -----");
2 // requete
3 String moyReq = "SELECT AVG(e.salaire) FROM employe e";
4 // execution + recup
5 double salaireMoy = (double)em.createQuery(moyReq).getSingleResult();
6 // affichage
7 System.out.println(" Salaire moyen : "+salaireMoy);

```

## Clauses

## 1. Between And

```

1 System.out.println("----- Clause BETWEEN / AND -----");
2 String betweenAndReq = "SELECT e FROM employe e"
3   + " WHERE e.salaire BETWEEN 30000 AND 32000 ";
4 // def de la requete + exec + recup
5 List<Employe> listeSalaires = em.createQuery(betweenAndReq).getResultList();
6 // affichage
7 System.out.println("Employe entre 30000 et 32000");
8 for (Employe employe :listeSalaires) {
9     System.out.println(employe.getNom()+"\t\t"+employe.getSalaire());
10 }

```

## 2. Like

```

1 System.out.println("----- Clause LIKE -----");
2 String likeReq = "SELECT e FROM employe e WHERE e.nom LIKE 'o%'";
3 // def de la requete + exec + recup
4 List<Employe> listeNoms = em.createQuery(likeReq).getResultList();
5 // affichage
6 System.out.println("Nom commancant par O ");
7 for (Employe employe :listeNoms) {
8     System.out.println(employe.getNom());
9 }

```

## Paramètres

Le passage de paramètres dans une requête peut s'effectuer de 2 façons :

- ? suivi du rang du paramètre
- : suivi du label du paramètre

Après création de la requête, il faut setter en utilisant `setParameter()` avec soit le rang soit le label.

## JDBC vs JPA

Connection ----- EntityManager

Statement ----- Query

PreparedStatement ----- NamedQuery

## 4.3 Annotations

Le mapping de nos classes java par le JPA, se réalisent par les annotations. Elles s'appliquent à chaque classe ainsi qu'à ses attributs.

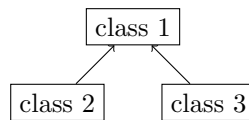
1. annoter la classe :

- `@Entity(name="...")` : Il est déconseillé de renommer manuellement les entités.  
A SAVOIR : Dans une requête SQL, Hibernate va donner le nom de la classe Java à l'entité en minuscule alors que JPA va donner le nom exact de votre classe à l'entité.
- `@Table(name="...")`  
Par défaut, la création d'une entité prend le nom de la classe. Les colonnes d'une table prend les noms de ses attributs. Par défaut, les Id ne sont pas auto-incrémentés.

2. annoter les attributs :

- `@Id` : signale une clé primaire
- `@GeneratedValue(strategy=GenerationType.IDENTITY)` : génération auto-incrémentiel
- `@Column(name="...")` : ajout de *nullable=false* pour les PK
- `@Transient` : signale de ne pas créer une colonne pour cette attribut

## 4.4 Héritage



Mapping JPA : 3 types d'implémentation. La stratégie par défaut est `SINGLE_TABLE_STRATEGY`. Le choix se définit par l'annotation `@Inheritance`.

1. `SINGLE_TABLE_STRATEGY`

La définition de la stratégie s'établit avec des annotations dans la classe mère et les classes filles.

(a) Classe ascendante avec PK :

- `@Entity`
- `@Table`
- `@Inheritance(strategy=InheritanceType.SINGLE_TABLE)` : génère 1 seule table pour les 3 entités
- `@DiscriminatorColumn(name="type_personnel")` :

(b) Classe descendante :

- `@Entity`
- `@DiscriminatorValue(value="PT")`

2. `JOINED_STRATEGY`

(a) Classe ascendante avec PK :

- `@Entity`
- `@Table`
- `@Inheritance(strategy=InheritanceType.JOINED)` : génère autant de tables que de classes
- `@DiscriminatorColumn(name="type_personnel")` :

(b) Classe descendante :

- `@Entity`
- `@Table`
- `@PrimaryKeyJoinColumn(referencedColumnName="id")` : lien avec la PK de la classe ascendante
- `@DiscriminatorValue(value="PT")`

## 3. TABLE\_PER\_CLASS\_STRATEGY

(a) Classe ascendante avec PK :

- @Entity
- @Table
- @Inheritance(strategy=InheritanceType.TABLE\_PER\_CLASS) : génère autant de tables que de classes

(b) Classe descendante :

- @Entity
- @Table
- Pas d'annotation pour l'id. On récupère la définition de la PK de la classe ascendante

## 4.5 Associations

Les associations impliquent des cardinalités / multiplicités. On aura la correspondance entre classe avec table, et association/cardinalité avec PK/FK.

1 – 1 : OneToOne

1 – \* : OneToMany

\* – 1 : ManyToOne

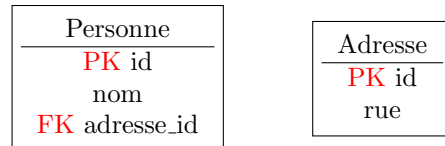
\* – \* : ManyToMany

## 4.5.1 OneToOne

UML



Java / BDD



Personne.java

```

1 @Entity
2 public class Personne {
3
4     @Id
5     @GeneratedValue(strategy=GenerationType.IDENTITY)
6     private int id;
7     private String nom;
8
9     @OneToOne() // one personne to one adresse
10    @JoinColumn(name="adresse_id",referencedColumnName="ID")
11    private Adresse adresse;
  
```

Adresse.java

```

1 @Entity
2 public class Adresse {
3
4     @Id
5     @GeneratedValue(strategy=GenerationType.IDENTITY)
6     private int id;
7     private String rue;
8     private String ville;
9
10    @OneToOne(mappedBy="adresse") // one adresse to one personne
11    @JoinColumn(name="personne_id",referencedColumnName="ID")
12    private Personne personne;
  
```

Dans cette configuration, la clé étrangère sera placée dans la table Personne.



### 4.5.2 OneToMany - ManyToOne



La clé étrangère est portée par le côté *Many* i.e \*. On utilise alors `@ManyToOne` avec `@JoinColumn`. Le côté non porteur doit être définie avec uniquement `@OneToMany(mappedBy="")`.

Le fait d'avoir un *Many* implique l'utilisation d'une liste.

Client.java

```

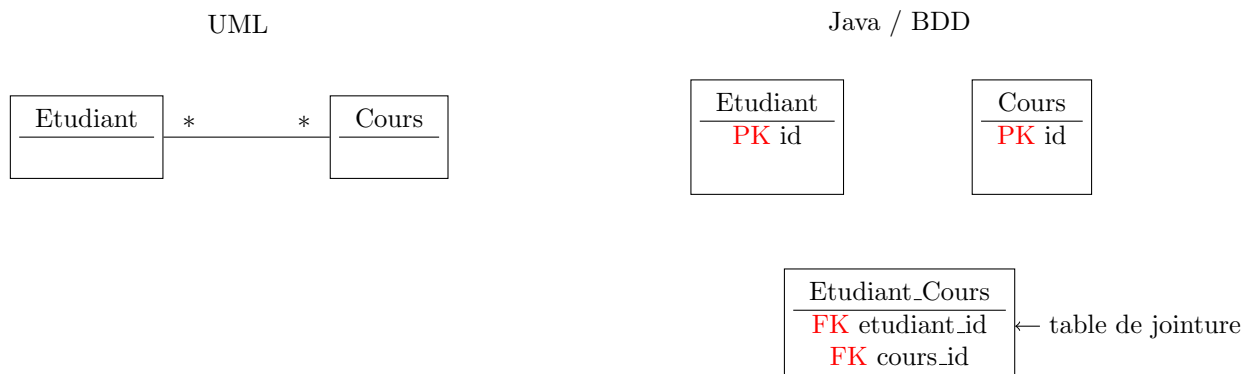
1  @Entity
2  public class Client {
3
4      @Id
5      @GeneratedValue(strategy=GenerationType.IDENTITY)
6      @Column(name="id_client")
7      private int id;
8      private String nom;
9      private String numero;
10
11      // one client to many commands
12      // cote non porteur de la FK -> mappedBy
13      @OneToMany(mappedBy="client")
14      private List<Commande> commandes;
  
```

Commande.java

```

1  @Entity
2  public class Commande {
3
4      @Id
5      @GeneratedValue(strategy=GenerationType.IDENTITY)
6      @Column(name="id_commande")
7      private int id;
8      @Temporal(TemporalType.TIMESTAMP)
9      private Date date;
10     private double totalPrix;
11
12     // many commands to one client
13     // cote porteur de la FK
14     @ManyToOne
15     @JoinColumn(name="client_id",referencedColumnName="id_client")
16     private Client client;
  
```

### 4.5.3 ManyToMany



Le *ManyToMany* s'utilise avec une table de jointure. Un coté définit la table de jointure avec `@JoinTable` et l'autre coté utilise la propriété `mappedBy` pour pointer sur cette table. On peut ajouter également la propriété `cascade` pour répercuter les modifications d'une table sur les tables liées. Le `mappedBy` fait en sorte que si une entité est supprimée, l'entité associée l'est aussi. Pour éviter cela, il ne faut pas spécifier de `mappedBy` et définir `@JoinTable` des deux côtés.

Etudiant.java

```

1  @Entity
2  public class Etudiant {
3
4      @Id
5      @GeneratedValue(strategy=GenerationType.IDENTITY)
6      @Column(name="id_etudiant")
7      private int id;
8      private String nom;
9
10     // many etudiant to many cours
11     @ManyToMany
12     @JoinTable(name="etudiant_cours",
13               joinColumns = @JoinColumn(name="etudiant_id"),
14               inverseJoinColumns = @JoinColumn(name="cours_id") )
15     private List<Cours> listeCours = new ArrayList<>();

```

Cours.java

```

1  @Entity
2  public class Cours {
3
4      @Id
5      @GeneratedValue(strategy=GenerationType.IDENTITY)
6      @Column(name="id_cours")
7      private int id;
8      private String nom;
9
10     // many cours to many etudiant
11     @ManyToMany(mappedBy="listeCours", cascade=CascadeType.ALL)
12     private List<Etudiant> listeEtudiant = new ArrayList<>();

```

#### 4.5.4 Eager et Lazy

Ceux sont des concepts d'accès à une entité d'une table. Dans le cas où plusieurs tables sont reliées avec différentes associations. A la sélection d'une entité, la stratégie **EAGER** va récupérer toute l'arborescence concernée par cette entité. La stratégie **LAZY** sélectionne uniquement l'entité sans prendre en compte les éventuelles dépendances dans les autres classes. Par défaut, Hibernate travaille en **LAZY** et JPA en **EAGER**.

## 4.6 DTO : Data Transfert Object

Le but est de créer une classe dédiée à la recherche par critère uniquement côté Java. Ses attributs sont les mêmes que la classe cible. L'utilité est d'écrire une méthode qui va tester les critères à sélectionner et ainsi ajuster le format à la requête initiale.

Nous créerons un objet de recherche vide. Nous settons uniquement les attributs sur lesquels nous voulons faire la recherche.

exemple

```

1  private String createQuery(UserDto userSearch) {
2      String pQuery = "SELECT u FROM User u WHERE 1=1"; // requete toujours satisfaite
3      if (userSearch.getLogin() != null) {
4          pQuery += " AND u.login='" + userSearch.getLogin() + "'";
5      }
6      if (userSearch.getPassword() != null) {
7          pQuery += " AND u.password='" + userSearch.getPassword() + "'";
8      }
9      if (userSearch.getVille() != null) {
10         pQuery += " AND u.ville='" + userSearch.getVille() + "'";

```

```
11     }
12     if (userSearch.getNom()!=null) {
13         pQuery += " AND u.nom='"+userSearch.getNom()+"'";
14     }
15     if (userSearch.getPrenom()!=null) {
16         pQuery += " AND u.prenom='"+userSearch.getPrenom()+"'";
17     }
18     if (userSearch.getDateNaissance()!=null) {
19         pQuery += " AND u.dateNaissance='"+userSearch.getDateNaissance()+"'";
20     }
21     return pQuery; // retourne la requete finale a effectuer
22 }
```



## Chapitre 5

# HTML : HyperText Markup Language

C'est un langage de balisage [1].

### 5.1 Déclaration

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Page Title</title> <!-- titre de la page -->
5 </head>
6 <body style="background:yellow">
7
8 <h1 style="color:red">This is a Heading</h1> <!-- titre -->
9 <h2 style="color:#11E899">This is a Heading</h2>
10 <h3>This is a Heading</h3>
11
12 <p>This is a paragraph.</p>
13
14 </body>
15 </html>
```

### 5.2 Mise en page

```
1 <body>
2     <header></header>
3     <nav></nav> <!-- flottant -->
4     <section>
5         <article></article>
6     </section>
7     <footer></footer>
8 </body>
```

Toutes les parties flottantes sont à déclarer avant quelconque partie fixe. Les styles de chaque partie est définie avec le CSS.

### 5.3 Style CSS

Le style est géré dans un fichier `.css` qui est ensuite appelé dans le `head` avec la balise suivante :

```
1 <link rel="stylesheet" href="style.css" />
```

#### 5.3.1 Attributs

- background

- padding
- margin
- width

## 5.4 JavaScript [2]

C'est un langage de programmation interprété ligne par ligne dans une balise *script*. Il est généralement géré dans un fichier *.js*. Il n'y a pas de variable *caractère* mais uniquement des chaînes de caractère. La déclaration du type de variable n'est explicitement écrit mais s'établit à la première instanciation. La méthode *alert* crée un pop-up avec notre message.

### 1. Demander une confirmation

```
1 if( confirm("executer ?") ) {
2     alert("hello");
3 }
```

### 2. Tableaux

```
1 var varTab = new Array('Sam','Angela','Selena',5);
2 varTab.push("Oneill_"); /* ajoute une valeur */
3 varTab.shift(); /* retire la 1ere valeur */
4 varTab.pop(); /* retire la derniere valeur */
5
6 for (var id in varTab) { /* afficher */
7     alert(varTab[id]);
8 }
```

### 3. Dictionnaire

```
1 var family = {self: 'Sebastien',
2               sister: 'Laurence',
3               brother: 'Ludovic',
4               cousin_1: 'Pauline',
5               cousin_2: 'Guillaume'};
```

### 4. Formulaire

Un formulaire se déclare avec la balise **form**. On peut y ajouter des boutons *valider* cependant une balise **button** va effacer le contenu de notre formulaire une fois activé, il faut donc utiliser une balise **input** avec l'attribut *type=button*.

### 5.4.1 DOM : Document Object Model

Cet outil est utilisé pour récupérer des objets de la page HTML et ainsi interagir avec l'utilisateur.

#### 1. getElementById

```
1 var div = document.getElementById("test").innerHTML;
2 alert(div);
```

#### 2. getElementsByTagName

```
1 var divs = document.getElementsByTagName('div');
2 for ( var id in divs ) {
3     alert("n "+divs[id].innerHTML);
4 };
```

#### 3. getElementsByName

#### 4. innerHTML

La méthode *get* récupère un objet. Il faut donc utiliser *innerHTML* pour avoir le corps de notre objet.

## Exemple d'affichage dans la page HTML

```

1 <body>
2 <button onclick="ho()">afficher</button>
3 <div id="myDiv"></div>
4
5 <script id="alert">
6     function ho() {
7         document.getElementById("myDiv").innerHTML = "hi !";
8     }
9 </script>
10 </body>

```

Le déclenchement d'une fonction peut se faire avec l'attribut *onclick*.

## 5.4.2 Fonctions

### 5.4.3

## 5.5 JQuery [3]

JQuery est une bibliothèque JavaScript avec des méthodes prêtes à l'emploi. L'utilité est d'optimiser le dynamisme des pages Web. C'est une extension de JavaScript avec une syntaxe différente.

### 5.5.1 Déclaration

Elle s'établit dans le **head** en indiquant le fichier `jquery.js` soit avec un emplacement local soit avec une adresse Web.

## exemple

```

1 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>

```

### 5.5.2 Utilisation

Le script se place dans une balise **script**. L'utilisation de jQuery s'effectue avec le symbole **\$** ou simplement avec le mot clé **jQuery**. Le DOM utilisé en JavaScript est remplacé par **\$(document)** pour sélectionner tout le document. Il est possible de sélectionner n'importe quelle balise de la même manière en indiquant entre guillemets le nom de la balise. Pour sélectionner par **id** plutôt que par balise, il faut le signaler avec le symbole **#**. Enfin, pour sélectionner par **class** plutôt que par balise, il faut le signaler avec le symbole **.**

Afin d'appliquer un traitement au document, il faut utiliser la méthode **ready**. Après seulement, il est possible de définir plusieurs fonctions à appliquer à différentes sélections.

## exemple

```

1 <head>
2 <script src="jquery-3.2.1.js"></script>
3 <script>
4     $(document).ready(function(){
5         $("button").click(function(){
6             alert("Attention");
7             $("#p1").hide();
8             $(this).hide();
9         });
10    });
11 </script>
12 </head>
13 <body>
14     <h2>Test 1</h2>
15     <p id="p1">Test 2</p>
16     <p id="p2">Test 3</p>
17     <button>Appliquer le cache</button>
18 </body>

```

Plusieurs `function()` peuvent être définies pour une même action.

### 5.5.3 Correspondance

Connection	-----	EntityManager
Statement	-----	Query
PrepareStatement	-----	NamedQuery

### 5.5.4 Fonctions

- hide()
- show()
- slideDown(), slideUp() : paramètre possible **slow**, **fast**
- slideToggle()
- animate() : paramètre possible entre accolades éventuellement avec un chiffre pour définir la vitesse.
- hover()

exemple

```
1 animate({left:'+=100',top:'+=100',width:'+=50',height:'+=50', opacity:'0.5'},1500);
```

L'opacité est réglée sur 1 par défaut.

## 5.6 BootStrap

BootStrap est un framework contenant des styles prédéfinies. Il était conçu à l'origine pour faciliter la création de page Web pour les développeurs. La contre-partie a été la baisse des effectifs de designers.

## Chapitre 6

# JSF : Java Server Faces

JSF est apparu pour la première fois le 11 Mars 2004. Plusieurs versions de JSF sont apparues, pour atteindre aujourd'hui la dernière version JSF 2.2 sorti en Avril 2013. C'est un framework Java destinée pour le Web. Il est basé sur une architecture MVC : Model, View, Controller.

- Model : représenté par des entités ou des JavaBeans.
- View : pages web au format jsp, jsf à l'origine. A partir de la v1.2 de JSF, on utilise des pages XHTML.
- Controller : représenté par la Faces servlet. Cette partie est prédéfinie dans un fichier XML.

Lorsque l'on parle de JSF, on parle de couche Web. La configuration se réalise par deux fichiers XML.

1. `web.xml` : toujours présent dans un projet Web
2. `faces-config.xml` : configuration de l'utilisation de JSF

### 6.1 Structures

### 6.2 Configuration

#### 6.2.1 `web.xml`

Le premier fichier descripteur de toute application web J2EE. Le fichier `web.xml` contenu dans le répertoire `WEB-INF`. Il décrit les 3 principaux éléments :

1. La page d'accueil de l'application : `welcome-file`
2. La servlet mère du JSF : `servlet-class`
3. Associer les vues portant l'extension `.xhtml` à la `FacesServlet` : `url-pattern`.

exemple

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xmlns="http://java.sun.com/xml/ns/javaee"
4     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
5     version="3.0">
6     <display-name>tpJSF</display-name>
7
8     <servlet>
9         <servlet-name>Faces Servlet</servlet-name>
10        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
11        <load-on-startup>1</load-on-startup>
12    </servlet>
13
14    <servlet-mapping>
15        <servlet-name>Faces Servlet</servlet-name>
16        <url-pattern>/faces/*</url-pattern>
17    </servlet-mapping>
18
19    <context-param>
20        <description>State saving method: 'client' or 'server' (=default). See JSF Specification 2.5.2</description>
```



```

21     <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
22     <param-value>client</param-value>
23 </context-param>
24
25 <context-param>
26     <param-name>javax.servlet.jsp.jstl.fmt.localizationContext</param-name>
27     <param-value>resources.application</param-value>
28 </context-param>
29
30 <listener>
31     <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
32 </listener>
33
34
35 <welcome-file-list> <!-- page d'accueil de l'appli -->
36     <welcome-file>faces/index.jsp</welcome-file>
37 </welcome-file-list>
38
39 </web-app>

```

### 6.2.2 faces-config.xml

Le fichier gérant la logique de l'application web s'appelle par défaut `faces-config.xml`. Il est placé dans le répertoire `WEB-INF` au même niveau que `web.xml`. La balise de départ est `<faces-config>`. Il décrit essentiellement six principaux éléments :

- `<managed-bean>` : Définit les managed Bean.
- `<navigation-rule>` : Définit les règles de navigation.
- `<message-bundle>` : Définit les ressources éventuelles.
- `<resource-bundle>` : Définit la configuration de la localisation.
- `<validator>` : Définit la configuration des validateurs.
- `<converter>` : Définit la configuration des convertisseurs.

Le fichier de configuration est un fichier XML décrit par une DTD.

exemple

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <faces-config
3     xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
6     version="2.2">
7
8     <navigation-rule> <!-- 1ere page afficher -->
9         <display-name>index</display-name>
10        <from-view-id>/index.xhtml</from-view-id>
11    </navigation-rule>
12
13 </faces-config>

```

### 6.2.3 ManagedBean

C'est une classe javaBean gérée par JSF. Elle doit respecter un ensemble de directives :

- Un constructeur sans arguments.
- Doit contenir des getteurs et des setteurs pour les attributs de la classe.

Elle peut être définie à partir du fichier `faces-config.xml` ou par le biais d'annotations `@ManagedBean`. Elle permet de faire le lien entre les interfaces utilisateurs et le code métier de l'application.

**Annotation** Il faut l'importer de la librairie `javax.faces.bean`.

### 6.2.4 Facelets

C'est une technologie de présentation pour le développement d'applications web en Java. Facelets est spécifiquement développé pour JSF, se sont des pages XHTML et contenant des balises propres à JSF, chargées respectivement d'afficher des données, formulaire de saisie...

```

1 <ui:composition xmlns="http://www.w3.org/1999/xhtml"
2   xmlns:h="http://java.sun.com/jsf/html"
3   xmlns:f="http://java.sun.com/jsf/core"
4   xmlns:ui="http://java.sun.com/jsf/facelets"
5   xmlns:p="http://primefaces.org/ui" template="WEB-INF/template.xhtml">

```

Dans une Facelet, une bibliothèque de balises est incluse via l'ajout d'un attribut `xmlns` à la balise `<html>` qui ouvre le corps de la page. Il s'agit là d'un namespace XML.

### 6.2.5 Unified Expression Language

JSF propose une syntaxe basée sur des expressions qui facilitent l'utilisation des valeurs d'un bean. Ces expressions doivent être délimitées par `#` et `.`. Une expression est composée du nom du bean suivi du nom de la propriété désirée séparés par un point.

exemple

```

1 <p:inputText id="email" value="#{etudiantBean.etudiant.email}">

```

la syntaxe utilisée par JSF est proche mais différente de celle proposée par JSTL : JSF utilise le délimiteur `#` ... et JSTL utilise le délimiteur `$` ... .

## 6.3 Le cycle de vie

Il se base sur 6 étapes et 4 événements. JSF commence par une demande de la page Web.

### 6.3.1 Interfaces graphiques

1. Composants de saisies :  
inputHidden inputSecret inputText inputTextArea selectBooleanCheckbox selectManyCheckbox selectManyListbox selectManyMenu selectOneListbox selectOneMenu selectOneRadio
2. Composants de sorties Column messages dataTable outputText outputFormat outputLink graphicImage
3. Composants de commandes commandButton commandLink
4. Composants de regroupements form panelGrid, panelGroup

### 6.3.2 Options additionnelles

## 6.4 Utilisation

Il est conseillé d'appliquer un `timeout` important pour éviter le redémarrage intempestif du server. Le `ManagedBean` permet de réaliser toutes les opérations CRUD. Il gère toutes les interfaces relatives à une entité. Pour naviguer entre les pages Web, on utilise des méthodes qui retournent les index des pages. La configuration des index se fait dans le fichier `faces-config.xml`.

## Chapitre 7

# HTTP : Hypertext Transfer Protocol

C'est un protocole de requête entre client et server. On peut donc faire des requêtes GET, POST, PUT, DELETE, HEAD. Il utilise le plus souvent le format JSON.

**JSON** est un protocole d'échange basé sur des clés et des valeurs

exemple

```
1 {"nom": "Lecomte", "prenom": "Samuel"}
```

```
1 {
2   "nom" : "Walid MELLOULI",
3   "jobs" : {
4     { "boite" : "Mecode Software", "mission" : "Consultant BigData", "date" : "2015" },
5     { "boite" : "Karavel/Promovacances", "mission" : "Chef de projet Technique", "date" : "2013" },
6     { "boite" : "Proxiad (SSII)", "mission" : "Ingenieur E&D Java/J2EE", "date" : "2011" },
7     { "boite" : "FitnetManager", "mission" : "Ingenieur E&D Java/J2EE", "date" : "2010" }
8   },
9   "contributions" : {
10    { "software" : "elasticsearch-hadoop", "url" : "https://github.com/elastic/elasticsearch-hadoop/pull/889" }
11  },
12  "certifications" : {
13    { "certif" : "Hadoop Platform and Application Framework", "date" : "2015" },
14    { "certif" : "Introduction to Functional Programming", "date" : "2015" },
15    { "certif" : "Functional Programming Principles in Scala", "date" : "2015" },
16    { "certif" : "MongoDB for Java Developers (M101J) / MongoDB for DBAs (M102)", "date" : "2014" },
17    { "certif" : "Java Enterprise Edition 5 Web Component Developer, Oracle Certified Professional", "date" : "2009" },
18    { "certif" : "Java Standard Edition 5 Programmer, Oracle Certified Professional", "date" : "2008" }
19  },
20  "email" : "mellouli.walid@gmail.com"
21 }
```

## 7.1 Service REST

Par convention, on a la correspondance suivante :

- GET : récupérer
- PUT : insert
- POST : mise à jour
- DELETE : suppression
- HEAD :

La méthode GET permet de transmettre des requêtes visibles sur le réseau avec une URI.

```
1 GET www.google.fr
2 GET www.google.fr/?gws_rd=ssl
```

interroger une ressource

```
1 GET ../directory
```

La réponse d'un GET vide sera une liste au format JSON ou csv suivant la demande du client.

recuperer une personne

```
1 GET ../directory/person/<person>
```

La réponse sera l'objet correspondant à la personne cherchée. On aura un code 200 Success si la ressource est trouvée ou un code 404 not found.

mettre à jour une personne

```
1 PUT ../directory/person/<person>
```

## 7.2 Code erreur

- 200 : succès de la requête ;
- 301 et 302 : redirection, respectivement permanente et temporaire ;
- 401 : utilisateur non authentifié ;
- 403 : accès refusé ;
- 404 : page non trouvée ;
- 500 et 503 : erreur coté serveur.
- [https://fr.wikipedia.org/wiki/Liste\\_des\\_codes\\_HTTP](https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP)

## 7.3 IP

Par défaut, l'adresse IP de notre localhost est 127.0.0.1 Le port 80 est le port par défaut pour les accès ver internet.



## Chapitre 8

# ElasticSearch

@Walid Mellouli

### 8.1 Introduction

#### 8.1.1 Qu'est qu'un moteur de recherche

- Moteur d'indexation des données
- Moteur de recherche des données dans l'index
- Indexation efficace des données : sur tout les champs
- Analyse des données :
- Recherche text (Tokenizing, Stemming, Filtering)
- GeoLocation
- Date parsing
- Score de pertinence (Relevance scoring)

#### 8.1.2 Token

Analyse de données : Tokenizing

En java

```
1 "Ceci_est_un_exemple_de_tokenisation".split("_")
```

=> Tokens : Ceci, est, un, exemple, de, tokenisation

#### 8.1.3 Base

C'est un moteur de recherche

- NoSQL orienté document (JSON)
- Basé sur Apache Lucene
- HTTP / REST / JSON
- Distribué (le coeur ElasticSearch est présent sur une machine et distribué sur les autres), Scalable (peu importe la quantité de données le temps de réponse est inchangé), Cloud ready
- Apache2 License
- Ecrit en Java (cross-plateform)

### 8.1.4 Historique

Elasticsearch a été créé par Shay Banon

- Première version publique 0.4.0, le 8 février 2010
- 2012 création de l'entreprise Elasticsearch par Shay Banon et Steven Schuurman (Amsterdam, Pays-Bas)
- La version 1.0 sort le 12 février 2014

#### Un moteur de recherche

- clusterisé
- répliqué
- API REST
- Facilement scalable
- Support de fonctionnalités de recherche avancées (Full Text)
- Orienté document (i.e. JSON)
- Configurable et extensible
- Communauté active

### 8.1.5 Apache Lucene

Bibliothèque open source écrite en Java

- Supporté par Apache Foundation
- Créé par Doug Cutting
- Hébergé en 2000 sous SourceForge
- Le coeur de plusieurs moteurs de recherche :
  - Elasticsearch
  - Solr
  - Nutch

Apport d'Elasticsearch à Lucene ?

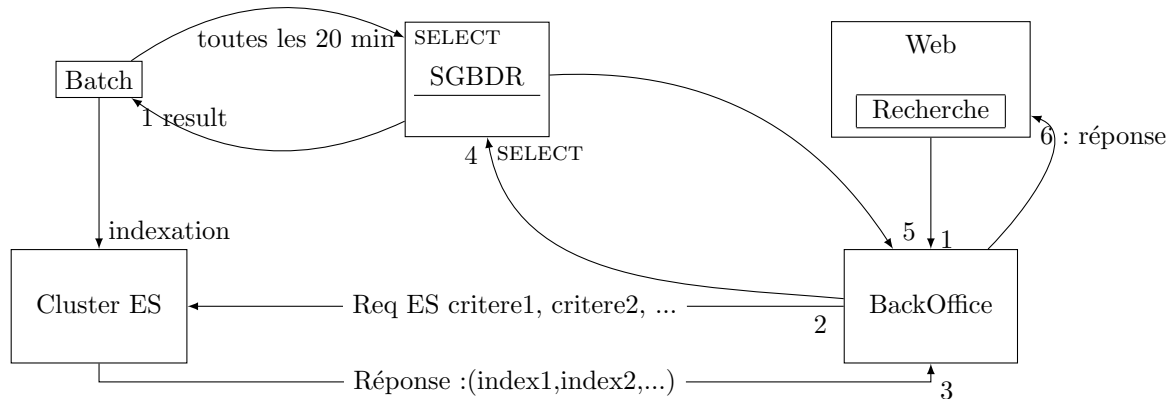
- Service RESTfull : JSON API over HTTP (interroger depuis n'importe quel plateforme)
- Distribué (Lucene est mono), Haute disponibilité (High Availability) et Performance
- Clustering
- Persistance à long terme (non recommandé, Elasticsearch est un moteur de recherche et non une base de données)
- Ecriture vers un système de stockage persistant

### 8.1.6 SGBDR vs Elasticsearch

Database	–	Index
Table	–	Type
Row	–	Document
Column	–	Field (Champs)
Schema	–	Mapping
Index	–	All field indexed
SQL	–	Query DSL
SELECT * FROM table ...	–	GET http ://...
UPDATE table SET ...	–	PUT http ://...

**Indexation** En SQL, la recherche par attributs devient trop longue sur des grandes quantités de données. De fait, il va scanner toutes les valeurs de notre attribut jusqu'à trouver la valeur voulue. Une solution est de créer une table d'index sur certains attributs pour au lieu de chercher dans la BDD, on cherche dans notre table d'index qui est liée aux données. La recherche sera plus rapide, cependant l'insertion et la suppression sera alourdie du fait de mettre à jours la ou les tables d'index. En règle générale à partir d'un certain volume, une base de données n'est pas adaptée pour la recherche.

### 8.1.7 Principe



## 8.2 Installation

**Tuto :** [https://www.elastic.co/guide/en/elasticsearch/reference/2.4/\\_installation.html](https://www.elastic.co/guide/en/elasticsearch/reference/2.4/_installation.html)

**More :** <https://www.elastic.co/guide/en/elasticsearch/reference/2.1/setup-service.html>

Prérequis :

- Java, il faut indiquer la variable d'environnement de Java sous le nom `JAVA_HOME`.
- Un client REST pour interroger l'api elasticsearch via des requêtes HTTP GET/POST/PUT/DELETE/HEAD
  - RESTClient (firefox)
  - REST Easy (firefox)
  - RESTED (firefox)
  - Postman (chrome)
  - Sense Recommandé
- Télécharger elasticsearch :
 

<https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/zip/elasticsearch/2.4.0/elasticsearch-2.4.0.zip>

## 8.3 Elasticsearch-head

Depuis le répertoire d'Elasticsearch, installer le plugin par la commande

```
1 .\bin\plugin install mobz/elasticsearch-head
```

Pour y accéder : `localhost:9200/_plugin/head/` Nous avons horizontalement les différents noeuds et verticalement les différents index créés.

## 8.4 Création/suppression

Ces méthodes sont utilisables à travers une API REST. Créer un index :

- Method : PUT
- URL : `http://localhost:9200/test-index/`

Supprimer l'index :

```
1 DELETE http://localhost:9200/test-index/
```

On peut monitorer le noeud elasticsearch en utilisant elasticsearch-head.

## 8.5 Distribué/haute disponibilité

Un cluster peut contenir plusieurs serveurs (noeuds). But :

- agir en tant que service unique
- gérer noeuds de données et noeuds clients

### 8.5.1 Sharding

- Les index sont constitués de plusieurs shards (le nombre de shards est configurable cf. 8.6).
- Chaque shard peut avoir zero ou plus de replicas
- les replicas se placent dans différents serveurs (server pools) afin d'assurer le basculement (failover) en cas de perte d'un noeud dans le cluster.

On a des shards primaires et des replicas (copie de la données). Elasticsearch ne place jamais un shard et son replica sur un même noeuds. Généralement, les shards sont sur les noeuds **Master** et les replicas sur un noeuds **Slave**.

### 8.5.2 Indexation

Lors de l'insertion d'une donnée, on a la création du shard et de son replica. A la suppression d'un shard, son replica est supprimé automatiquement.

### 8.5.3 Master

- Détection automatique du Master + failover (Basculement)
- Responsable de la distribution/équilibre des shards

## 8.6 Configuration

Elasticsearch est configurable par `elasticsearch.yml`.

```
1 # nom du cluster
2 cluster.name: hello-cluster
3 # le nom du noeud
4 node.name: hello-node
5 # HTTP port (9200 par défaut)
6 http.port: 9200
7 # le nom du rack/datacenter
8 node.rack: rack1
9 # nombre de shards (5 par défaut)
10 index.number_of_shards: 5
11 # nombre de replicas (1 par défaut)
12 index.number_of_replicas :1
```

## 8.7 Type de données

Elasticsearch supporte plusieurs types de données pour les champs des documents

- Type de données de base :
  - string, date, boolean, binary
  - long, integer, short, byte, double, float
- Type de données complexe : array, object, nested
- Type de données de Geolocation : geo\_point, geo\_shape
- Autres : ip (IPv4), completion, murmur3,...



## 8.8 Mappings/Settings

Création d'un index :

```
1 PUT http://localhost:9200/transactions
```

avec un body prédéfini :

```
1 {
2   "settings": {
3     "number_of_shards": 3,
4     "number_of_replicas": 1
5   },
6   "mappings": {
7     "scan": /* type de mapping */
8     "_source": {
9       "enabled": true /* sauvegarder la source d'indexation */
10    },
11    "_all": {
12      "enabled": false /* par défaut: true */
13    },
14    "dynamic": "false", /* tout autre attribut de propriétés sera ignore */
15    "properties": {
16      "scanId": {
17        "type": "string",
18        "index": "not_analyzed"
19      },
20      "storeType": {
21        "type": "string",
22        "index": "not_analyzed"
23      },
24      "productName": {
25        "type": "string",
26        "index": "not_analyzed"
27      },
28      "sector": {
29        "type": "string",
30        "index": "not_analyzed"
31      },
32      "quantity": {
33        "type": "integer",
34        "index": "not_analyzed"
35      },
36      "totalPrice": {
37        "type": "double",
38        "index": "not_analyzed"
39      }
40    }
41  }
42 }
43 }
```

voir les settings

```
1 GET http://localhost:9200/transactions/_settings
```

voir le mapping

```
1 GET http://localhost:9200/transactions/_mappings
```

Indexer un premier document :

```
1 PUT http://localhost:9200/transactions/scan/1
2 {
3   "scanId": "1",
4   "productName": "tee-shirt bleu",
5   "storeType": "Hypermarche",
6   "sector": "Textile",
7   "quantity": 3,
8   "totalPrice": 30.60
9 }
```

fermer un index

```
1 POST http://localhost:9200/transactions/_close
```

ouvrir un index

```
1 GET http://localhost:9200/transactions/_open
```

Recherche d'un document indexé

```
1 GET http://localhost:9200/transactions/scan/1
```

Mise à jours partielle d'un document

```
1 POST http://localhost:9200/transactions/scan/1/_update
2 {
3     doc : {
4         "mon_champ": "ma_nouvelle_valeur"
5     }
}
```

Suppression d'un document indexé

```
1 DELETE http://localhost:9200/transactions/scan/1
```

## 8.9 Batch API : \_bulk

Interroger elasticsearch par plusieurs d'opérations

```
1 POST http://localhost:9200/_bulk
2 /* premiere instruction */
3 {"index" : {"_index" : "transactions", "_type" : "scan", "_id" : "3" }}
4 /* document associe */
5 {"scanId":"3","productName":"jean levis","storeType":"Hypermarche","sector":"Textile","quantity":2,"totalPrice":
6     70.00}
7 /* deuxieme instruction */
8 {"delete" : {"_index" : "transactions", "_type" : "scan", "_id" : "6" }}
9 /* troisieme instruction */
10 {"update" : {"_index" : "transactions", "_type" : "scan", "_id" : "1" }}
11 /* document associe */
12 {"doc" : {"productName" : "Test Name"}}
```

L'utilité est de ne pas multiplier les connexions au serveur et ainsi gagner au temps de latence sur le réseau.

## 8.10 API de recherche : \_search

Chercher tous les documents dans tous les index

```
1 POST http://localhost:9200/_search
2 {
3     "query": {"match_all": {}}
4 }
```

Chercher tous les documents dans l'index transactions

```
1 POST http://localhost:9200/transactions/_search
2 {
3     "query": {"match_all": {}}
4 }
```

Pagination

```
1 POST http://localhost:9200/_search
2 {
3     "from" :0,
4     "size" :2,
5     "query": {
6         "match_all": {}
7     }
8 }
```

Sort (Tri)

```

1 POST http://localhost:9200/transactions/scan/_search
2 {
3     "sort": [
4         {"totalPrice": {"order": "desc"}}
5     ]
6     , "query": {"match_all": {}}
7 }

```

## 8.11 API \_cat

Affiche la santé du cluster

```
1 GET http://localhost:9200/_cat/health
```

Affiche les indices

```
1 GET http://localhost:9200/_cat/indices
```

Affiche les plugins installés

```
1 GET http://localhost:9200/_cat/plugins
```

Affiche les noeuds

```
1 GET http://localhost:9200/_cat/nodes
```

Affiche les shards

```
1 GET http://localhost:9200/_cat/shards
```

## 8.12 Routing

Lors d'une indexation, Elasticsearch répartit les documents indexés parmi les shards disponibles. Lors d'une recherche d'un document, Elasticsearch va interroger tous les shards et après il va agréger la réponse de tous avant de transmettre la réponse finale à l'utilisateur. Le routing est le fait d'indiquer comment et où placer les documents indexés. Lors de la création d'un index, il est faut ajouter une propriété **routing** dans le **mapping**.

voir 8.8

```

1 {
2     "mappings": {
3         "test": {
4             "_routing": {
5                 "required": true /* demandera un routing pour tous docs indexés */
6             }
7         }
8     }
9 }

```

Lors de l'indexation d'un document, on spécifie alors le routage à effectuer. De son côté Elasticsearch lance un `hash(routing1)%nb_shard`. De cette manière, tous les documents indexés avec le même routage dans ses propriétés, Elasticsearch les placera dans le même shard.

Indexation avec routing

```

1 PUT test-routing/test/1?routing=user1
2 {
3     "title": "This is a document"
4 }

```

Recherche avec routing

```
1 GET test-routing/test/1?routing=route1
```

## 8.13 \_dynamic

```

1 "mappings": {
2   "store": {
3     "_source": {
4       "enabled": true
5     },
6     "_all": {
7       "enabled": false
8     },
9     "dynamic": true,
10    "properties":

```

Le dynamisme permet d'indexer n'importe quel document sans spécifier au préalable les champs. Elasticsearch va indexer tous les champs du document indexé.

## 8.14 \_source

```

1 "_source": {
2   "enabled": true

```

Déclarer `source` avec `true` réalise une sauvegarde du document source indexé. C'est le comportement par défaut. Tous les champs du document seront alors visible lors d'une recherche avec un `match_all` par exemple. En spécifiant `false`, il faut ajouter à chaque champs de notre document la propriété `store:true`.

exemple

```

1 "properties": {
2   "storeId": {
3     "type": "integer",
4     "index": "not_analyzed",
5     "store": true
6   },
7   "name": {
8     "type": "string",
9     "index": "analyzed",
10    "analyzer": "store_analyzer"
11  },

```

Une recherche n'affichera que les champs qui ont été enregistrés.

## 8.15 Query/Filter

- Query : une clause `query` répond à la question « Dans quelle mesure ce document correspond-il à cette requête ? ». Une note est calculée.
- Filter : une clause `filter` répond à la question « Est-ce que ce document correspond à cette requête ? ». La réponse est un simple oui ou non, aucune note n'est calculée

Filter term

```

1 POST stores/store/_search
2 {
3   "query": {
4     "bool": {
5       "filter": {
6         "term": { "town": "LYON" }
7       }
8     }
9   }
10 }

```

Filter terms

```

1 POST stores/store/_search
2 {

```

```

3      "query": {
4        "bool": {
5          "filter": {
6            "terms": {"town": ["LYON", "PARIS"]} }
7          }
8        }
9      }
10    }

```

Should query / Scoring

```

1  POST stores/store/_search
2  {
3    "query": {
4      "bool" : {
5        "should" : [
6          {"terms" : {"postCode" : ["69003", "75018"]} },
7          {"term" : {"town" : "LYON"} }
8        ],
9        "minimum_should_match" : 1
10      }
11    }
12  }

```

## 8.16 Full-Text

Lors de l'indexation d'un document, les champs peuvent être analysés avant d'être indexés. L'analyser est configurable dans les settings de l'index.

settings

```

1  "analysis": {
2    "analyzer": {
3      "store_analyzer": { /* nom de notre analyser */
4        "type": "custom",
5        "tokenizer": "whitespace", /* mode de separation */
6        "filter": "lowercase" /* filtre a appliquer avant d'indexer */
7      }
8    }
9  }

```

mappings

```

1  "properties": {
2    "storeId": {
3      "type": "integer",
4      "index": "not_analyzed"
5    },
6    "name": {
7      "type": "string",
8      "index": "analyzed",
9      "analyzer": "store_analyzer"
10   },

```

## 8.17 GéoLocalisation

Afficher les données à partir d'un point GPS

```

1  POST stores/store/_search
2  {
3    "query": {
4      "filtered": {
5        "filter": {
6          "geo_distance": {
7            "distance": "4km",
8            "geoLocation": {

```

```

9         "lat": 45.747569,
10        "lon": 4.875327
11    }
12 }
13 }
14 }
15 }
16 }

```

Trier et afficher la distance par rapport à un point GPS

```

1 POST stores/store/_search
2 {
3   "query": {
4     "filtered": {
5       "filter": {
6         "geo_distance": {
7           "distance": "4km",
8           "geoLocation": {
9             "lat": 45.747569,
10            "lon": 4.875327
11          }
12       }
13     }
14   }, "sort": [
15     {
16       "_geo_distance": {
17         "geoLocation": {
18           "lat": 45.747569,
19           "lon": 4.875327
20         },
21       },
22       "order": "asc",
23       "unit": "m"
24     }
25   ]
26 }
27 }

```

## 8.18 Agrégations

- Agréger des données suite à une requête de recherche
- Une agrégation peut être considérée comme une unité de travail qui établit des informations analytiques sur un ensemble de documents.
- Basé sur des blocs de construction appelés « aggregation ». Les « aggregations » peuvent être composés =, des résumés complexes des données.
- Le contexte de l'exécution (query/filter) définit le jeu de documents
- Il existe de nombreux types d'agrégations :
  - Bucketing : création des « buckets »
  - Metric : calculer des métriques sur un ensemble de documents

### 8.18.1 Agrégations imbriquées

- Les agrégations peuvent être imbriquées
- Les agrégations « Bucketing » peuvent avoir des sous-agrégations de « Bucketing » ou « Metric »
- Les sous-agrégations seront calculées à partir des « Buckets » générés par l'agrégation parent
- Pas de limite de profondeur des agrégations imbriquées

### 8.18.2 Metrics aggregations

- avg : calculer la moyenne

- sum : calculer la somme
- max : calculer le max
- min : calculer le min
- cardinality : nombre de valeur distinct

Moyenne du prix sur tous les documents

```

1 POST transactions/scan/_search
2 {
3     "size":0, /* n'affiche pas de document */
4     "aggs" :{
5         "avg_totalPrice" :{"avg" :{"field" : "totalPrice" }}
6     }
7 }

```

Par défaut, sans spécifier une `query`, la recherche s'effectue sur tous les documents.

### 8.18.3 Bucket aggregations

- Les « bucket aggregations » ne calculent pas les métriques sur des champs
- Ils créent des buckets (groupe) de documents. Chaque bucket est associé à un critère selon le type d'agrégation
- Ils calculent également le nombre de documents qui "tombent dans" chaque bucket
- Par opposition aux Metrics Aggregations, les « Bucket Aggregations » peuvent contenir des sous-agrégations

Terms Aggregation

```

1 POST transactions/scan/_search
2 {
3     "size": 0,
4     "aggs": {
5         "groupBy_storeType": {
6             "terms": {
7                 "field": "storeType"
8             }
9         }
10    }
11 }

```

Range Aggregation

```

1 POST transactions/scan/_search
2 {
3     "size": 0,
4     "aggs": {
5         "quantities_ranges": {
6             "range": {
7                 "field": "quantity",
8                 "ranges": [
9                     {
10                        "from": 0,
11                        "to": 3
12                    },
13                    {
14                        "from": 3,
15                        "to": 5
16                    },
17                    {
18                        "from": 5
19                    }
20                ]
21            }
22        }
23    }
24 }

```

Calculer le chiffre d'affaires et la moyenne de vendus par type de magasin

```

1 POST transactions/scan/_search
2 {
3   "size": 0,
4   "aggs" :{
5     "groupBy_storeType" :{
6       "terms" :{"field": "storeType" },
7       "aggs" :{
8         "chiffre" :{"sum" :{"field" : "totalPrice" }},
9         "quantite" :{"avg" :{"field" : "quantity" }}
10      }
11    }
12  }
13 }

```

Calculer le chiffre d'affaires et la moyenne des quantités totale vendues par type de magasin « storeType » et rayon « sector »

```

1 POST transactions/scan/_search
2 {
3   "size": 0,
4   "aggs" :{
5     "groupBy_storeType" :{
6       "terms" :{"field": "storeType" },
7       "aggs" :{
8         "groupBy_sector" :{
9           "terms" :{"field" : "sector" },
10          "aggs" :{
11            "chiffre" :{"sum" :{"field" : "totalPrice" }},
12            "quantite" :{"avg" :{"field" : "quantity" }}
13          }
14        }
15      }
16    }
17  }
18 }

```

## 8.19 Java project

Import Maven projet. Clic droit -> Maven -> Update : pour mettre à jour les dépendances





## Chapitre 9

# Kafka

@Samy Dindane

### 9.1 Intro

Kafka est un agent de message distribué créé par LinkedIn. Il est écrit en scala et java. Il a été conçu pour :

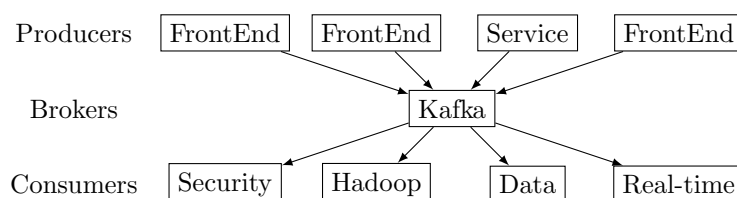
- être résistant
- les grosses volumétries
- le traitement temps réel
- la scalabilité
- la faible latence

#### 9.1.1 Role

##### Historique :

1. architecture spaghetti : tout est désordonné
2. architecture en couche : impact trop important sur les autres couches si changement d'un service d'une couche. Ce système Monolithe accède à une seule base de données.
3. architecture micro-service : les codes sont ainsi isolés et permettent le changement sans engendrer des bugs dans tous les services. L'impact est limité.

Kafka permet de découpler les pipelines de données. Au lieu d'établir une connexion entre un service de récupération de données et un service de persistance, chacun est connecté à Kafka et il distribue efficacement l'information.



#### 9.1.2 Utilisation

- système de messages
- analytique web
- indicateur de suivi
- traitement en flux
- Agrégation de logs
- Commit log

### 9.1.3 Stream

### 9.1.4 Terminologie

1. **topics** : les données sont entreposées dans les topics. On conserve généralement les données d'un même type dans un même topic.
  - Les topics sont répartis sur plusieurs partitions. L'information est ainsi partagée sur plusieurs machines.
  - Chaque partition est une séquence ordonnée et immuable de messages.
  - Les messages reçoivent un ID séquentiel appelé offset.
  - La donnée est retenue pour une période configurable.
  - Le nombre de partition peut être augmenté après la création d'un topic.
  - Les partitions sont assignées à un broker.

Résumé : Les partitions servent à diviser les flux d'informations des topics pour passer par différents brokers. On a ainsi une gestion de la panne.

2. **brokers** : ce sont les serveurs sur lesquels sont stockés les topics et qui font tourner les instances de Kafka.
  - Service tournant au sein du cluster Kafka
  - Reçoit les messages des producers et les rend disponibles au consommateurs
  - Coordonné grâce à Zookeeper
  - Stocke les messages sur le système de fichier
  - Réplique les messages de et vers d'autres brokers
  - Fourni les informations de métadonnées à propos des brokers, topics et partitions
  - Depuis Kafka 0.9.0 – gère la coordination des consommateurs.
3. **replicas** : duplication des données.
  - Les partitions dans un topic doivent être répliquées
  - Chaque partition a 1 leader et 0 ou plusieurs followers
  - Un In-Sync Replica (ISR) est un replication qui communique avec Zookeeper et qui rattrape le plus vite possible le leader
  - Le facteur de réplication peut être augmenté après la création mais pas diminué

NB : Ainsi, une ou plusieurs partitions sont dédiées à la réplication de données au lieu du transport de données.

4. **producer** : publication de messages dans Kafka. Les producers transmettent les données aux brokers.
  - Les producers publient les messages vers un topic
  - À la publication, ils distribuent eux-mêmes les messages entre les partitions : Round-robin et Key hashing
  - Ils envoient les messages assignés à une partition de manière synchrone ou asynchrone vers le broker qui est le leader de cette partition.  
ACKS = 0 (none), 1 (leader), -1 (all ISRs)
5. **consumer** : écoute des messages. Les consumers reçoivent les données des brokers.
  - Lisent les messages d'un topic
  - Plusieurs consommateurs peuvent lire le même topic
  - Ils gèrent leurs offsets (position) de consommation
  - Les messages restent dans Kafka après qu'ils soient consommés. Cela dépend de la stratégie adoptée cf 9.2.2.

### 9.1.5 ZooKeeper

Gestion de la configuration distribué. Kafka utilise ZooKeeper pour sauvegarder sa configuration.

### 9.1.6 TP Lab1

#### Install Kafka & Setup

<https://kafka.apache.org/documentation.html#quickstart>

Téléchargement de Kafka

```
1 wget http://apache.mindstudios.com/kafka/0.10.0.0/kafka_2.11-0.10.0.0.tgz
2 tar xvf kafka_2.11-0.10.0.0.tgz
3 mv kafka_2.11-0.10.0.0 kafka
```

Kafka se trouve maintenant de le répertoire Kafka. Pour fonctionner, il faut installer au préalable.

Démarrage de zookeeper (dans le terminal 1)

```
1 kafka/bin/zookeeper-server-start.sh kafka/config/zookeeper.properties
```

Démarrage du server Kafka (sur un autre terminal)

```
1 kafka/bin/kafka-server-start.sh kafka/config/server.properties
```

Très bien, le server Kafka tourne! Vous pouvez laisser le terminal 2 de coté.

Création des topics Kafka (terminal 3)

```
1 kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1
  --topic clickstream
```

Vérification de la création du topic (terminal 3)

```
1 kafka/bin/kafka-topics.sh --list --zookeeper localhost:2181
```

Description du topic

```
1 kafka/bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic clickstream
```

#### Test de Kafka

Maintenant que Kafka est configuré, nous allons envoyer quelques messages.

Démarrage d'un producteur en console (terminal3)

```
1 kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic clickstream
```

Maintenant tout ce que vous allez taper dans le terminal 3 est stocké dans Kafka (ligne à ligne). Démarrage d'un consommateur en console (terminal 4)

Dans un nouveau terminal start the consumer application

```
1 kafka/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic clickstream --from-beginning
```

A la fin vous devriez avec une installation qui ressemble à ça : Vous pouvez taper des lignes le terminal 3 et regarder ce qui se passe dans le terminal 4.

Exemple de ligne à taper

```
1 hello world
2 i am alive
```

Une fois que vous avez observé le comportement du consommateur, vous pouvez le relancer (CtrlC pour l'arrêter, et relancer le programme). regardez ce qui se passe.

## Gestion des partitions

Augmentez le nombre de partition du topic clickstream

```
1 kafka/bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic clickstream --partition 2
```

2. Vérifiez que le topic a bien 2 partitions (voir ligne de commande plus haut) Topic :clickstream PartitionCount :2 ReplicationFactor :1 Configs : Topic :clickstream Partition :0 Leader :0 Replicas :0 Isr :0 Topic :clickstream Partition :1 Leader :0 Replicas :0 Isr :0

Lister les enregistrement de la partition 2 uniquement

```
1 kafka/bin/kafka-simple-consumer-shell.sh --broker-list localhost:9092 --offset -2 --partition 1
--print-offsets --topic clickstream
```

## 9.2 Architecture

### 9.2.1 Consumer groups

Le but est de définir à Kafka un ensemble de consumers travaillant ensemble. Ainsi, Kafka :

- repartit la charge entre les instances
- gère les erreurs en équilibrant les partitions
- sauvegarde les offset

### 9.2.2 Stratégie

Un consumer est en trois étapes :

1. demande à Kafka de lui fournir un job. Il peut lire de la donnée
2. traite la donnée
3. envoie une confirmation de traitement

Ces étapes peuvent être définies suivant deux stratégie :

- AtMostOne : chaque job sera réalisé une seule fois.
  1. le consumer demande à Kafka de la données
  2. Kafka donne un job avec un offset et l'oublie en considérant que le travail est fait
  3. traitement
  4. crash éventuel : la donnée est perdu
- AtLeastOne : chaque job doit être traité au moins une fois.
  1. le consumer demande à Kafka de la données
  2. Kafka donne un job avec un offset et attend la confirmation de traitement
  3. traitement
  4. envoie de confirmation ou crash
  5. Kafka envoie le job à un autre consumer s'il n'a pas de confirmation.

### 9.2.3 Garanties

Les messages envoyés d'un producer vers une partition spécifique d'un topic sont ajoutés à la partition dans l'ordre dans lequel ils sont envoyés. Une instance de consumer voit les messages dans l'ordre où ils sont stockés dans le log. Pour un topic avec un facteur de réplication de N, nous allons pouvoir tolérer N-1 serveur en erreur sans perdre un des messages sauvegardés dans le log. L'ordre des messages est sauvegardé par partition mais pas par topic.

## 9.3 Développer avec Kafka

- Python – kafka-python / pykafka
- Go – sarama / go\_kafka\_client
- C/C++ - librdkafka / libkafka
- .NET – kafka-net / rdakafka-dotnet
- Node.js – kafka-node / sutoiku/node-kafka
- HTTP – kafka-pixy / kafka-rest

### 9.3.1 API Producer

```

1 // configure
2 Properties config = new Properties();
3 config.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
4 KafkaProducer producer = new KafkaProducer(config);
5 // create and send a record
6 ProducerRecord record = new ProducerRecord("topic", "key".getBytes(), "value".getBytes());
7 Future<RecordMetadata> response = producer.send(record); // this is always non-blocking
8 System.out.println("the offset was; " + response.get().offset()); // get() blocks

```

client.id	identifie l'application du producer
producer.type	async ou sync
request.required.acks	ackling semantics
serializer.class	cf. plus loin

producer.java

```

1 package fr.univalence.kafkatraining.testapi;
2
3 import java.util.Properties;
4 import org.apache.kafka.clients.producer.ProducerRecord;
5 import org.apache.kafka.clients.producer.KafkaProducer;
6
7 class TestKafkaProducer {
8     private static Properties kafkaProps() {
9         Properties props = new Properties();
10        props.put("bootstrap.servers", "localhost:9092");
11        props.put("acks", "all");
12        props.put("retries", 0);
13        props.put("batch.size", 16384);
14        props.put("linger.ms", 1);
15        props.put("buffer.memory", 33554432);
16        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
17        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
18        return props;
19    }
20    public static void main(String[] args) {
21        KafkaProducer<String, String> producer = new KafkaProducer<String, String>(kafkaProps());
22        producer.send(new ProducerRecord<String, String>("clickstream", null, "somepayload"));
23        System.out.printf("sent data");
24        producer.close();
25    }
26 }

```

#### Producer asynchrone

- Envoi des messages non-bloquants
- Permet de batcher
- Utilise un pool de producteur synchrones
- Communique en utilisant une queue
- Utilise une copie de la configuration du producteur async

### 9.3.2 L’Ack d’un message

#### Concept :

- Le message est commité quand «le nombre d’ISR requis» l’a reçu
- L’ack est l’accusé de réception renvoyé au broker
- «Le nombre d’ISR requis» est défini par `request.required.acks`

Seul les producteurs doivent s’en occuper. Cela permet d’équilibrer entre latence (vitesse) et longévité (sécurité des données)

#### Valeurs pour `request.required.acks` :

- 0 : aucune attente (meilleure latence)
- 1 : attend l’ack du leader
- 1 : attend l’ack de tous les ISR (meilleure durabilité)

Attention à `request.timeout.ms` !

### 9.3.3 API Consumer

producer.java

```

1 package fr.univalence.kafkatraining.testapi;
2
3 import org.apache.kafka.clients.consumer.ConsumerRecord;
4 import org.apache.kafka.clients.consumer.ConsumerRecords;
5 import org.apache.kafka.clients.consumer.KafkaConsumer;
6 import org.apache.kafka.common.PartitionInfo;
7 import org.apache.kafka.common.TopicPartition;
8 import java.util.ArrayList;
9 import java.util.Arrays;
10 import java.util.List;
11 import java.util.Properties;
12
13 class TestKafkaConsumer {
14     private static Properties kafkaProps() {
15         Properties props = new Properties(TestKafkaProducer.kafkaProps());
16         props.put("bootstrap.servers", "localhost:9092");
17         props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
18         props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
19         return props;
20     }
21     public static void main(String[] args) {
22         KafkaConsumer<String,String> consumer = new KafkaConsumer<String,String>(kafkaProps());
23
24         // Gestion automatique des inscriptions aux partitions
25         // consumer.subscribe(Arrays.asList("clickstream"))
26
27         // Gestion manuelle des inscriptions aux partitions
28         // Recuperation de la liste des partitions du topic clickstream
29         List<TopicPartition> topicPartitions = new ArrayList<TopicPartition>();
30         for (PartitionInfo partitionInfo : consumer.partitionsFor("clickstream")) {
31             topicPartitions.add(new TopicPartition(partitionInfo.topic(), partitionInfo.partition()));
32         }
33         //assignation de ce consumer a ces partitions
34         consumer.assign(topicPartitions);
35         consumer.poll(0);
36         //Rechargement depuis le debut
37         consumer.seekToBeginning(topicPartitions);
38         while (true) {
39             ConsumerRecords<String, String> records = consumer.poll(100);
40             for (ConsumerRecord<String, String> record : records) {
41                 System.out.println(record);
42             }
43             consumer.commitSync();
44         }
45     }
46 }

```

## 9.4 Zero Copy

Zero Copy pour les producers et les consumers de et depuis le broker cf. <http://kafka.apache.org/documentation.html#maximizingefficiency>. Les messages restent sur le disque quand ils sont consommés, la suppression se passe bien plus tard via le TTL ou la compaction cf. <https://kafka.apache.org/documentation.html#compaction>

## 9.5 Serializer et Deserializer

Le principe est de transformer une classe en un objet transportable et compréhensible par n'importe quel programme. Un serializer va donc concaténer les attributs de notre objet en une chaîne de caractères avec un séparateur pour qu'il soit compréhensible par Kafka. Le deserializer est utilisé pour faire l'inverse. Depuis une chaîne de caractères, il reconstruit notre objet avec ses attributs.

### Serializer

```

1 public class EventSerializer implements Serializer<Event> {
2     public byte[] serialize(String s, Event e){
3         // methode toString() pour concatener les attributs
4         String str = e.toString();
5         return str.getBytes(Charset.defaultCharset());
6     }
7 }

```

### Deserializer

```

1 public class EventDeserializer implements Deserializer<Event> {
2     public Event deserialize(String s, byte[] bytes) {
3         try {
4             String[] srtTab = new String(bytes).split(",");
5             Event e = new Event(
6                 Long.parseLong(srtTab[0]),
7                 srtTab[1],
8                 Integer.parseInt(srtTab[2]),
9                 srtTab[3],
10                srtTab[4],
11                Integer.parseInt(srtTab[5]),
12                Integer.parseInt(srtTab[6]));
13             return e;
14         } catch (Exception e) {
15             return null;
16         }
17     }
18 }

```

```

1 // changement dans producer.java
2 props.put("value.serializer",EventSerializer.class.getName());
3 // changement dans consumer.java
4 props.put("value.deserializer",EventDeserializer.class.getName());

```

### 9.5.1 Librairie Jackson

Librairie de serializer et deserializer en JSON.

#### pom.xml

```

1 <dependency>
2     <groupId>org.codehaus.jackson</groupId>
3     <artifactId>jackson-mapper-asl</artifactId>
4     <version>1.9.13</version>
5 </dependency>

```

Il a besoin d'un constructeur vide et de setteurs.

Le JSON est compatible avec la quasi-totalité des système. Le seul inconvénient est le temps de traitement sur de grosse volumétrie. Si l'on doit gérer plus de 10000 records/sec il est préférable d'utiliser le langage Avro.

## 9.6 Confluent

### 9.6.1 Kafka connect

Il fait partie de la plateforme Confluent créée en 2014 par les créateurs d'Apache Kafka chez LinkedIn.

- copie de données

### 9.6.2 Kafka Streams

- permet de faire du traitement
- très léger

Avec le traitement en flux, des problèmes de gestion arrivent. Kafka Stream est une alternative au framework de gestion de flux tel que Sparks, Flink, ...

## 9.7 Utilisation d'un connecteur Kafka Connect

En tant que développeur ou développeuse Data, il vous sera souvent demandé de récupérer des données depuis une source, et de les stocker dans un format et/ou dans un système différent. Le but de ce lab est de lire des données depuis Kafka et les stocker dans une base MySQL "à la main". Vous pouvez utiliser vos acquis comme JPA.

L'étape d'après est de faire la même chose avec avec connecteur Kafka Connect et de comparer les deux approches.

### 9.7.1 Étape 1

Reprendre le consumer écrit dans le lab 2 et mettre en place l'écriture dans une base de données de votre choix.

Installer MySQL

```
1 sudo apt-get install mysql-server mysql-client
```

Dépendences pour JPA

```
1 <dependency>
2   <groupId>org.hibernate.javax.persistence</groupId>
3   <artifactId>hibernate-jpa-2.0-api</artifactId>
4   <version>1.0.1.Final</version>
5 </dependency>
6 <dependency>
7   <groupId>org.hibernate</groupId>
8   <artifactId>hibernate-entitymanager</artifactId>
9   <version>5.2.10.Final</version>
10 </dependency>
11 <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
12 <dependency>
13   <groupId>mysql</groupId>
14   <artifactId>mysql-connector-java</artifactId>
15   <version>5.1.6</version>
16 </dependency>
```

### 9.7.2 Étape 2

Pour faire plus simple, on va utiliser une base de données SQLite pour lire des données, puis les injecter dans un topic Kafka.

Télécharger Confluent Platform depuis <http://packages.confluent.io/archive/3.2/confluent-3.2.0-2.11.tar.gz>. Créer un nouveau répertoire, vous pouvez l'appeler lab4-kafka-connect-sqlite par exemple. Copier les fichiers dans votre répertoire de travail.



```
1 $CONFLUENT_DIR/etc/schema-registry/connect-avro-standalone.properties
2 $CONFLUENT_DIR/etc/kafka-connect-jdbc/source-quickstart-sqlite.properties
```

Lancez la commande

```
1 sqlite3 test.db
```

Exécutez ces trois commandes

```
1 sqlite> CREATE TABLE accounts(id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, name VARCHAR(255));
2 sqlite> INSERT INTO accounts(name) VALUES('alice');
3 sqlite> INSERT INTO accounts(name) VALUES('bob');
```

Lancez le schema

```
1 $CONFLUENT_DIR/bin/schema-registry-start $CONFLUENT_DIR/etc/schema-registry/schema-registry.properties
```

Lancez un consumer de votre topic en exécutant

```
1 $CONFLUENT_DIR/bin/kafka-avro-console-consumer --new-consumer --bootstrap-server localhost:9092
2 --topic test-sqlite-jdbc-accounts --from-beginning
```

Dans un autre terminal lancez le connecteur

```
1 $CONFLUENT_DIR/bin/connect-standalone connect-avro-standalone.properties source-quickstart-sqlite.properties
```

Dans votre terminal de consumer, vous pouvez maintenant voir que les lignes que vous aviez inséré dans l'étape 4 sont disponibles dans votre topic.

Ajoutez une nouvelle ligne dans la base de données et notez ce qui se passe dans votre topic.

## 9.8 Kafka en python

cf. 10.12

## 9.9 Kafka avec spark

cf. 20.1

## 9.10 Autres

changer la mémoire accessible pour Kafka

```
1 KAFKA_HEAP_OPT='-Xmx500M -Xms500M'
```

### Test de connaissances Kafka

1. Qu'est ce qu'un microservice ?
2. Quels sont les éléments principaux de Confluent Platform ? À quoi sert celui qu'on a vu en lab ?
3. Pourquoi JSON est un choix intéressant pour le transport de données ?
4. Pourquoi fait-t-on du streaming ?
5. Quel concept utilise Kafka pour être tolérant à la faute ?
6. Donnez des cas d'usage d'une plateforme streaming
7. Donnez quelques inconvénients d'un système streaming
8. Qu'est ce qu'un broker ?
9. Comment faire pour que plusieurs instances d'une application puissent lire un stream Kafka en même temps sans conflit ?
10. Plusieurs applications différentes peuvent-elles lire le même stream Kafka en même temps ?



# Chapitre 10

# Python

@Nicolas Geniteau

## 10.1 Introduction

### Caractéristique

- Multi-plateformes (Windows, Linux, MacOS, etc.)
- Programmation multi-paradigmes (impérative, objet, fonctionnelle)
- Interprété
- Typage dynamique

### Productivité

- Lisible
- Consis
- Ecosystème (scientifique, intelligence artificielle, maintenance, big data, etc.) Outils (notebook, virtualenv, etc.)

### Utilisation

- Prototypages
- Scripts
- Applications graphiques
- Web

**Convention PEP** La norme veut une indentation de 4 espaces.

### Env

- python
- ipython
- notebook

## 10.2 Environnement virtuel

Création d'un environnement

```
1 python3 -m venv venv
```

## Charger un environnement

```
1 source venv/bin/activate
```

## Installer un notebook

```
1 pip install wheel
2 pip install notebook
```

## Exemple de script

```
1 #!/usr/bin/env python
2 import sys
3 import math
4 def fibo(n):
5     phi = (1+math.sqrt(5))/2
6     phiprim = -1/phi
7     return (phi**n-hiprim**n)/math.sqrt(5)
8 if __name__ == '__main__':
9     print(int(fibo(int(sys.argv[1]))))
```

## 10.2.1 Matplotlib

## Paquets à installer

```
1 apt-get install libpng-dev python3-dev libfreetype6-dev libxft-dev
```

## Dans notre venv

```
1 pip install matplotlib
```

## 10.2.2 REST

## Dans notre venv

```
1 pip install requests
```

## exemple de requête

```
1 > API_VELOV_URL = "https://download.data.grandlyon.com/ws/rdata/jcd_jcdecaux.jcdvelov/all.json"
2 # execution de la requete
3 > requests.get(API_VELOV_URL).text
4 # execution de la requete + conversion en dict avec JSON
5 > json.loads(requests.get(API_VELOV_URL).text)["values"]
```

## exemple avec paramètre

```
1 > API_URL = "https://maps.googleapis.com/maps/api/geocode/json"
2 > param = {'address' : '54 rue des rancy 69003' }
3 > res = json.loads(requests.get("{}?{}".format(API_URL,urlencode(param))).text)
```

## 10.3 Programmation fonctionnelle

- lambda
- map
- filter
- functools.reduce
- (list—map—etc) comprehension

### 10.3.1 lambda

```
1 > (lambda x: x + 1)(10)
2 11
```

### 10.3.2 map

```
1 > test = [1,2,3,4]
2 > def inc(x):
3 ...     return x+1
4 > list(map(inc, test))
5 > list(map(lambda x: x+1, test))
6 > list(map(lambda x: (x,x*x), test))
```

### 10.3.3 filter

```
1 > list(filter(lambda x: x%2 == 0, test))
```

### 10.3.4 reduce

```
1 > from functools import reduce
2 > test = [1,2,3,4]
3 > reduce(lambda x,y :x+y, test)
4 10
```

### 10.3.5 list comprehension

```
1 > list([x+1 for x in l])
2 [2,3,4,5]
3 > list([(x,x*x) for x in l])
4 [(1, 1), (2, 4), (3, 9), (4, 16)]
5 > list([(x,x*x) for x in l if x%2==0])
6 [(2, 4), (4, 16)]
```

```
1 > x = [0,3,8,11]
2 > print(x[0::2])
3 [0,8]
4 > print(x[1::2])
5 [3,11]
```

## 10.4 Exception

Générer une exception

```
1 raise Exception
```

```
1 try:
2     d['truck']
3 except:
4     print("exception")
```

```
1 try:
2     d['truck']
3 except KeyError as e:
4     print("exception")
5     print(e)
6 finally:
7     print("fin")
```

## 10.5 Classes

### Exemple

```

1 class Animal(object):
2     _couleur = ""
3     _name = ""
4     def __init__(self, name=""):
5         print("Hi")
6         self._name = name
7     def walk(self):
8         print("marche")
9     def get_name(self):
10        return self._name
11 class Alien(object):
12     def fly(self):
13         print("fly")

```

### Hérédité

```

1 class Dog(Animal):
2     def bark(self):
3         print("abolement")
4     def walk(self):
5         #super(Dog, self).walk()
6         print("run")

```

### Hérédité multiple

```

1 class SuperDog(Dog, Alien):
2     pass

```

## 10.6 Générateur

Un générateur est un outils d'itération.

```

1 def my_range(n):
2     i = 0
3     while i < n:
4         yield i
5         i += 1

```

Le mot clé `yield` agit comme un `return`. Il retourne la valeur de `i` calculé et sort de la boucle. Toutefois, à la prochaine commande de `my_range`, il reprend à l'endroit où l'exécution s'était arrêtée et continue la boucle. Ce système permet d'itérer sur de grande plage sans saturer la mémoire étant donné que l'on crée une seule itération à chaque étape et non toute la liste comme le fait la commande `range(n)`.

## 10.7 Géolocalisation

### 10.7.1 Obtenir coordonnées GPS

#### module requis

```

1 pip install geolocation-python

```

#### exemple

```

1 from geolocation.main import GoogleMaps
2 def address(pAddress):
3     google_maps = GoogleMaps(api_key = 'your_google_maps_key')
4     location = google_maps.search(location = pAddress).first()
5     return (location.lat, location.lng)

```

### 10.7.2 Calcul de distance

module requis

```
1 pip install haversine
```

Haversine permet également d'obtenir des coordonnées GPS cf. Rosettacode.org.

## 10.8 TP Velov

script final

```
1  #!/usr/bin/env python
2  # coding: utf-8
3  import requests
4  import json
5  from urllib.parse import urlencode
6  from haversine import haversine
7  import sys
8  from tabulate import tabulate
9  API_URL = "https://maps.googleapis.com/maps/api/geocode/json"
10 API_VELOV_URL = "https://download.data.grandlyon.com/ws/rdata/jcd_jcdecaux.jcdvelov/all.json"
11 def getPostion(address):
12     param = {'address': address}
13     res = json.loads(requests.get("{}?{}".format(API_URL, urlencode(param))).text)
14     location = res['results'][0]['geometry']['location']
15     return float(location['lat']), float(location['lng'])
16 def get_stations():
17     return json.loads(requests.get(API_VELOV_URL).text)['values']
18 def add_distance(stations, pos):
19     for s in stations:
20         pos_station = (float(s["lat"]), float(s["lng"]))
21         s["distance"] = haversine(pos, pos_station)
22 def sort_stations(stations):
23     return sorted(stations, key=lambda x: x["distance"])
24 def get_closest_stations(address):
25     stations = get_stations()
26     add_distance(stations, getPostion(address))
27     return sort_stations(stations)[:10]
28 def main():
29     address = sys.argv[1]
30     stations = get_closest_stations(address)
31     res = []
32     for s in stations:
33         res.append([s["address"], s["available_bikes"], s["available_bike_stands"]])
34         #print("{} - {} - {}".format(s["address"], s["available_bikes"], s["available_bike_stands"]))
35     print(tabulate(res, headers=("Adresse", "Vélos libres", "Places restantes")))
36 if __name__ == "__main__":
37     main()
```

## 10.9 TP Tweeter

<https://python-twitter.readthedocs.io/en/latest/>

module tweepy

```
1 pip install tweepy
```

Certaines APIs nécessitent une authentification. Le protocole auth est donc utilisé.

script final

```
1  #!/usr/bin/env python
2  # coding: utf-8
3  from tweepy import Stream
4  from tweepy import OAuthHandler
5  from tweepy.streaming import StreamListener
6  import json
```

```

7 import sys
8
9 consumerKey = "3Ax78wxvjlrboTUXx5eHnWayU"
10 consumerSecret = "GVo6WDH1ay3vhuAeosICCsKQ1nV2VQN62ZZagqUV6DLRSn7Ah"
11 accessTokenKey = "860356285191028736-Dw0iGkvW1LGocxRweFnpS9Xt1BNLEzM"
12 accessTokenSecret = "JqeLKXi3Ej2o6dPhYiXAJVRmASRQc73P710DKvkMFfEmb"
13
14 class listener(StreamListener):
15     def __init__(self, filename):
16         self.fid = open(filename, "w")
17     def on_data(self, data):
18         all_data = json.loads(data)
19         tweet = all_data["text"]
20         username = all_data["user"]["screen_name"]
21 # self.fid.write("{}"+username+"{}{}"+tweet+"{}\\n")
22 self.fid.write(username+" : "+tweet+"\\n")
23 self.fid.flush()
24 return(True)
25     def on_error(self, status):
26         print(status)
27 def main(filename, key):
28     auth = OAuthHandler(consumerKey, consumerSecret)
29     auth.set_access_token(accessTokenKey, accessTokenSecret)
30     twitterStream = Stream(auth, listener(filename))
31     twitterStream.filter(track=[key], async=True)
32 if __name__ == "__main__":
33     try:
34         key = sys.argv[1]
35         dest = "/tmp/tweets.json"
36         print("save to {}".format(dest))
37         main(key, dest)
38     except IndexError:
39         print("Indiquer un mot clef")
40     except:
41         print("Aucune connexion internet")

```

## 10.10 Git

### Créer un dépôt Git

```

1 git init
2 git config --global user.email "you@example.com"
3 git config --global user.name "Your Name"

```

### Ajouter un lien vers notre script

```

1 git add velov.py
2 git commit -s

```

L'option `commit -s` permet d'écrire la trace de notre ajout

### Différentes commandes

```

1 git status
2 git log

```

### Affiche ce qu'on a installer

```

1 pip freeze
2 pip freeze > requirements.txt
3 // ajouter les requis a notre depot
4 git add requirements.txt
5 git commit -s

```

```

1 vim .gitignore
2 git add .gitignore
3 git commit -s

```

## Générer une clé SSH

```

1 ssh-keygen
2 /* genere une cle prive id_rsa et publique id_rsa.pub*/
3 /* copier la cle publique*/
4 cat id_rsa.pub

```

Il est nécessaire de récupérer l'adresse de notre dépôt GitHub en utilisant ssh.

Pour envoyer sur le serveur

```

1 /* lien avec notre depot distant*/
2 git remote add origin git@github.com:seansaoo/inti.git
3 /* envoie de notre projet*/
4 git push --set-upstream origin master

```

## 10.11 Crawler/Scraper

Un crawler sert à récupérer des informations des sites internet. Un scraper sert à récupérer uniquement la data, généralement de sites qui ne fournissent pas d'API.

**Tutorial :** <https://doc.scrapy.org/en/latest/intro/tutorial.html>

**More :** <https://doc.scrapy.org/en/latest/topics/spiders.html>

Paquets requis

```
1 sudo apt-get install libxml2-dev libxslt1-dev zlib1g-dev libffi-dev libssl-dev
```

```
1 pip install scrapy
```

Création d'un projet

```
1 scrapy startproject nom_du_projet
```

'blogs\_spider.py' dans le dossier spiders

```

1 import scrapy
2 class QuotesSpider(scrapy.Spider):
3     name = "blogs"
4     start_urls = ["http://blog.python.org/"]
5     def parse(self, response):
6         for quote in response.css("div.post.hentry"):
7             yield {
8                 "title": quote.css("h3.post-title.entry-title a::text").extract_first(),
9                 "author": quote.css("span.fn::text").extract_first(),
10                "url": quote.css("h3.post-title.entry-title a::attr(href)").extract_first()
11            }
12        nextPage = response.css("a.blog-pager-older-link::attr(href)").extract_first()
13        if nextPage is not None:
14            nextPage = response.urljoin(nextPage)
15            yield scrapy.Request(nextPage, self.parse)

```

Lancement de scrapy

```

1 scrapy crawl blogs
2 # avec enregistrement en JSON
3 scrapy crawl blogs -o results.json

```

**Recherche récursive** Pour permettre la recherche récursivement sur plusieurs pages, il faut indiquer la valeur `False` dans le fichier `settings` du projet.

settings.py

```

1 # Obey robots.txt rules
2 ROBOTSTXT_OBEY = False

```



## 10.12 Kafka

### Kafka Producer

```
1 from kafka import KafkaProducer
2 producer = KafkaProducer(bootstrap_servers="localhost:9092")
3 producer.send("foobar", b"some_message_bytes")
```

send line per line from file

```
1 #!/usr/bin/env python3
2 # coding: utf-8
3 from kafka import KafkaProducer
4 import json
5 producer = KafkaProducer(bootstrap_servers="localhost:9092")
6 file = open("pop-test/src/main/resources/raw_pop.json")
7 for i in range(100000):
8     msg = json.loads(file.readline().strip())
9     producer.send("jsonstream", json.dumps(msg).encode("utf-8"))
```

### Kafka Consumer

```
1 from kafka import KafkaConsumer
2 import json
3 consumer = KafkaConsumer(bootstrap_servers="localhost:9092")
4 consumer.subscribe(["topic"])
5 for msg in consumer:
6     temp = json.loads(msg.value.decode("utf-8"))
7     print(temp)
```

## 10.13 Script Shell

```
1 #!/bin/sh
2 command
3 python script.py $@
```



# Chapitre 11

## Scala

@Adrien Broussolle

### 11.1 Introduction

#### L'Histoire

- Lambda calculus
- 1959 Lisp
- 1990 Erlang (premier langage fait par Ericson) and Haskell (langage universitaire)
- 2000 OCaml (universitaire) : 1er langage mélangeant Objet et Fonctionnel.
- 2003 Scala : créé à l'école d'informatique de Lausanne.
- 2005 F-Sharp : création de Microsoft (reprise d'OCaml avec la compatibilité Microsoft).
- 2014 Swift : création d'Apple.
- BigData, Hadoop
- Sémantique
- Twitter, LinkedIn : gestion de messages
- Netflix : gestion de vidéos
- Morgan Stanley, UBS, Barclays : transactions bancaires, modèle mathématiques d'investissement
- Apple, Samsung

#### Scala

- multi-paradigme
- compatible Java/JavaScript : on peut choisir la compilation Java ou JavaScript et ainsi utiliser toutes les bibliothèques existantes.
- typage statique inféré i.e le compilateur vérifie le typage avant l'exécution. *Inféré* signifie une sorte de typage dynamique à la définition d'une variable.
- peut-être le successeur de Java ? Java crée de nouveaux paquets déjà existants en Scala
- Financé par l'UE (2,3M€)

#### Installation

```
1 echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list
2 sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2EE0EA64E40A89B84B2DF73499E82A75642AC823
3 sudo apt-get update
4 sudo apt-get install sbt
```

**NB :** Ctrl+/ pour commenter

Lancement de l'interpréteur REPL

```
1 sbt consoleQuick
```

## POO

```
1 abstract class Animal{}
2
3 class Dog(name: String) extends Animal{
4     def bark = println("ouaf")
5     val description = "A dog named " + name + "."
6 }
7
8 val aDog = new Dog("medor")
```

Le constructeur est le corps de la classe

## 11.2 Core concepts

**Impératif VS Fonctionnelle** Un programme répond à une problématique métier.

- on n'explique pas : on exprime
- pas de variables : on utilise des valeurs `val`. Les valeurs sont immutables. Elles sont établies une seule fois et ne changent pas.
- pas de boucles : on utilise des fonctions `def`

Ni Python, ni Java, ni C++ n'exprime la fonction racine carrée.

### 11.2.1 Recursion

cf. Introduction à l'algorithmie

Rekursivité non-terminale

```
1 def factorial(n: Int) :Int = n match {
2     case x if x == 0=> 1
3     case x => x*factorial(x-1)
4 }
```

`factorail(42)` fait planter le programme. À chaque étape, le programme garde en mémoire la valeur et exécute `factorial(x-1)`. On arrive à la saturation de la mémoire.

En Scala, `@tailrec` permet d'indiquer un avertissement lors de la compilation s'il croit que la récursivité non-terminale. La récursivité non-terminale apparaît dès que l'on réalise une opération dans le retour `=>` et qui oblige l'OS à stocker de l'information.

**Accumulateur** contient les valeur déjà calculé.

Rekursivité terminale

```
1 def factorialAcc(n: Int, acc: Int) :Int = n match {
2     case x if x == 0=> acc
3     case x => factorialAcc(x-1, acc*x)
4 }
```

Cependant, le prototype est modifié. On peut alors utiliser une fonction auxiliaire qui appelle `factorialAcc`.

### 11.2.2 fonction d'ordre supérieur

Tout est fonction, implique que les paramètres sont des fonctions

```

1 > def sum(f: Int => Int, x: Int, y: Int) = {
2   f(x) + f(y)
3 }
4 > sum(fac, 3,4)

```

Fonction anonyme

```

1 > sum({x => x}, 3,4)

```

### 11.2.3 list

Création

```

1 > (1 to 5).toList
2 > List(1,2,3,4)
3 > List() ou Nil

```

- head renvoie le premier élément de la liste.
- :: pour concaténer des listes. Il est également utilisé pour pattern matcher sur une liste.

Exemple

```

1 > (1::Nil).head
2 1
3 > val test = List(('a',1),('b',2),('c',3))
4 > test(1)
5 ('b',2)
6 > test(1)._1
7 'b'
8 > test(1)._2
9 2

```

### 11.2.4 match

Exemple

```

1 def last(liste: List[Int]): Int = liste match {
2   case fin::Nil => fin
3   case fin::suite => last(suite)
4 }

```

### 11.2.5 map/filter

Exemple

```

1 def times(liste: List[Int]) = liste.map{x => x*2}
2 def paire(liste: List[Int]) = liste.filter{x => x%2==0}
3 def test(liste: List[Int]) = liste.map{x => x*3}.filter{x => x%2 ==0}

1 class Student(val followedSubjects: List[String])
2 case class Schools(bachelorStudents: List[Student], masterStudents: List[Student])
3
4 def triMasters(listSchool: List[Schools]) = listSchool.flatMap{x => x.masterStudents}
5 def triStudents(listSchool: List[Schools]) = listSchool.flatMap{x => x.masterStudents}
6   .filter{x => x.followedSubjects.exists(y => y=="Mecha")}

```

Lorsqu'une classe ne sert qu'à porter/contenir de la données, il est possible de la déclarer en tant que `case class`. Toutes ses attributs sont alors pris comme des valeurs et l'instanciation d'un objet de cette classe n'a pas besoin du mot clé `new`.

**Note :** Pour simplifier l'écriture, toutes les fonctions anonymes simples i.e. `x => x`, peuvent être remplacés par le symbole `_`.

```
1 def triStudents(listSchool: List[Schools]) = listSchool.flatMap{_.masterStudents}
2   .filter{_.followedSubjects.exists(_=="Mecha")}
```

sorted

```
1 def sortStudents(students: List[Student]) = students
2   .sortWith((x,y) => x.followedSubjects.count(_=="Physique") > y.followedSubjects.count(_=="Physique"))
```

### 11.2.6 fold

Prototype de *reduce*

```
1 List[T].foldLeft(accu:U){operation(accu:U, element:T) => U}
2
3 > List(1,2,3,4).foldLeft(0){(accu, element) => accu + element}
4 10
```

## 11.3 Scala concepts

### 11.3.1 sealed

Le mot clé **sealed** indique que la classe va être héritée. De plus, la classe **sealed** contiendra une énumération de classe seulement si celles-ci sont définies dans le même fichier unique.

### 11.3.2 Traits

Exemple

```
1 sealed trait Expr{
2   def eval: Float = this match {
3     case Num(n) => n
4     case Sum(e1, e2) => e1.eval + e2.eval
5     case Diff(e1, e2) => e1.eval - e2.eval
6     case Prod(e1, e2) => e1.eval * e2.eval
7     case Div(e1, e2) => e1.eval / e2.eval
8     case Abs(e1) => if (e1.eval<0) -1*e1.eval else e1.eval
9   }
10 }
```

Nous avons une énumération de cas suivant l'opération voulue.

Utilisation

```
1 case class Num(n: Float) extends Expr
2 case class Sum(e1: Expr, e2: Expr) extends Expr
3 case class Diff(e1: Expr, e2: Expr) extends Expr
4 case class Prod(e1: Expr, e2: Expr) extends Expr
5 case class Div(e1: Expr, e2: Expr) extends Expr
6 case class Abs(e1: Expr) extends Expr
```

Un trait n'est pas utiliser pour les services métiers.

## 11.4 Theorical

### 11.4.1 Impératif - Procédural

effet de bords = modification de l'état d'un objet et il y a des conséquences sur les autres services. C'est une source de bug.

À partir des années 70, on a ainsi créé le principe de scope qui rend les variables locales au lieu de globales jusqu'alors. Ensuite, on a encapsulé les variables dans des objets (i.e. Java). Enfin, on a introduit la visibilité (`private` ou `public`). Pour finir, il y a eu le Garbage Collector.

### 11.4.2 Lazy evaluation

**Fonctionnelle pure :** Appelée avec les mêmes paramètres une fonction pure retourne toujours la même valeur. Bonne pratique Meilleur couverture de test Limites les effets de bords Optimisation Exemple de fonctions impure ? toutes fonctions ayant des entrées/sortie avec l'extérieur

```

1 def eval = {
2     val x = {print("x"); 1} // affiche x et prend la valeur 1
3     lazy val y = {print("y"); 2} // attend d'être appelé
4     def z = {print("z"); 3} // attend d'être appelé
5     z + x + y + z + x + y // affiche xyz et calcule 1+3+2+3+1+2
6 }
7 > xyz
8 res:12

```

### 11.4.3 Pour et contre

plus de variable mutable

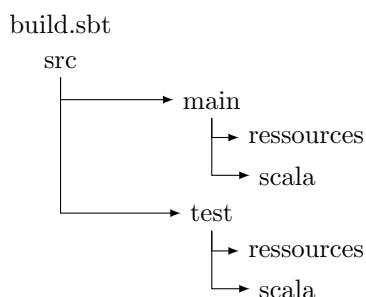
- Calcul parallèle
- Scalable/montée en charge
- Lazy evaluation
- Limite les effets de bords

Code élégant et concis

le code exprime la réalité métier et technique

- Maintenable, robuste
- Productif

## 11.5 SBT : Simple Build Tool



- `sbt run` : compile et lance le main
- `sbt compile` : compile le programme
- `sbt consoleQuick` : lance l'interpréteur Scala
- `sbt clean` : supprime les fichiers temporaires
- `sbt test` : lance le test défini dans notre projet

build.sbt

```

1 name := "Test Project"
2
3 version := "1.0"

```

```

4
5 scalaVersion := "2.11.8"
6
7 libraryDependencies += "org.scalatest" %% "scalatest" %% "2.2.6"

```

Les lignes blanches sont indispensables. On peut ajouter autant de librairies que nécessaire.

Exemple de test

```

1 class ExampleSpec extends FaltSpec with MustMatchers {
2   "A stack" must "pop values in last-in-first-out order" in {
3     val stack = new Stack[Int]
4     stack.push(1)
5     stack.push(2)
6     stack.pop() must be 2
7     stack.pop() must be 1
8   }

```

## 11.6 JSON

Dépendance

```

1 libraryDependencies += "com.typesafe.play" %% "play-json" % "2.6.0-M1"

```

Serializer et deserializer

```

1 case class Personne(name: String, age: Int, isMale: Boolean, address: Address)
2 case class Address(nb: Int, street: String, zip: String, town: String)
3 val address1 = Address(63, "route de la forge", "01100", "Oyonnax")
4 val angela = Personne("Angela", 27, false, address1)
5
6 implicit val personneReads = Json.reads[Personne]
7 implicit val personneWrites = Json.writes[Personne]
8
9 Json.toJson(samuel)
10
11 // si implicit n'est pas écrit il faut signaler le serializer
12 Json.toJson(samuel)(personneReads)

```

le fomart contient le writes et le reads

```

1 implicit val residentFormat = Json.format[Resident]

```

transformer un objet en JSON

```

1 Json.toJson(angela)

```

transformer du JSON en un objet

```

1 val selena = Json.parse("""
2   {
3     "name" : "Selena",
4     "age" : 1,
5     "isMale" : false,
6     "address" : {
7       "nb" : 63,
8       "street" : "route de la forge",
9       "zip" : "01100",
10      "town" : "Oyonnax"
11    }
12  }
13 """).as[Personne]

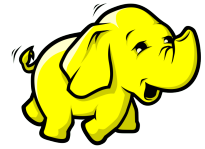
```

à la place de .as

```

1 Json.parse("""...""").validate[Annuaire] match {
2   case error: JsError => null
3   case JsSuccess(annuaire, _) => annuaire
4 }

```



## Chapitre 12

# Hadoop : High-Availability Distributed Object-Oriented Platform

@Walid Mellouli

### 12.1 Intro

#### Pourquoi Hadoop ?

- problème du stockage et de l'analyse des données
- capacité de stockage augmente mais le temps de lecture également
- nécessité de paralléliser les traitements en stockant sur plusieurs unités de DD
- la solution est la duplication des données comme le ferait un système RAID
- Système de sauvegarde de données distribué
- Hadoop fournit :
  - un système de fichiers HDFS
  - un système d'analyse de données : MapReduce

#### Histoire :

- créé par Doug Cutting (nutch, lucene) = limites de montée de charge
- 2003 : Google publie son système de fichier GFS
- 2004 : Google publie son système MapReduce
- 2006 : Hadoop est intégré à Apache Lucene
- 2008 : création du projet Hadoop

#### Cas d'utilisation

- Hadoop Use Cases
- Utilisation dans un cluster de plusieurs machines ( sur mono-machine pour les tests)
- Utilisation sur de grosse volumétrie

#### Yahoo

- un des plus grands utilisateurs depuis 2006
- 100,000 CPUs sur 40,000 serveurs Hadoop
- Clusters de 4,500 nœuds
- 455 petabytes de données ( 1 petabytes = 1000 terabytes )



## Ecosystème

### 12.1.1 Installation

- installer la version proposée par Apache
- installer une distribution basée sur le framework Hadoop :
  - Cloudera
  - Hortonworks
  - MapReduce

#### Cloudera VM

- Télécharger la VM de type VirtualBox sur [www.cloudera.com](http://www.cloudera.com)
- Configurer la VM avant le lancement (Allouer 5 Go de RAM, et augmenter la mémoire graphique)
- Lancer la VM Cloudera
- Configurer votre machine (affichage, clavier Azerty,...)
- Lancer l'onglet HUE

## 12.2 Yarn : Yet Another Resource Negotiator

- Un gestionnaire centralisé des ressources qui harmonise l'exploitation des ressources système Hadoop par les applications
- Des agents du gestionnaire de noeuds qui surveillent les opérations de traitement effectuées sur chaque noeud d'un cluster

## 12.3 HDFS : Hadoop Distributed File System

- HDFS reprend de nombreux concepts proposés par des systèmes de fichiers classiques : ext2 pour Linux, FAT pour Windows...
- On retrouve :
  - notion de blocs
  - les métadonnées
  - les droits
  - l'arborescence des répertoires

#### HDFS VS Systèmes de fichiers classiques

- Peut être déployé sur différents systèmes d'exploitation : il suffit d'avoir java
- Un système distribué : pas de limite, pour augmenter le volume global il suffit d'ajouter un nouveau noeud
- Tailles de blocs importante : 64 Mo, 128 Mo, 256 Mo, 512 Mo, ... (généralement 4 Ko dans un système classique)
- Système de réplication configurable

#### HDFS Architecture

##### HDFS NameNode

- Serveur Master
- Un NameNode gère :
  - l'arborescence des répertoires
  - les droits d'accès
  - ouverture, fermeture et renommage d'un fichier ou répertoire
  - le mapping des blocks dans les DataNodes

## HDFS DataNodes

- Serveurs Slaves / Workers
- Un DataNode gère :
  - le stockage de données
  - la lecture et l'écriture des requêtes client
  - création, suppression, réplication (demandé par NameNode)

## HDFS objectifs

- Éviter la défaillance des disques dur
- Traiter de volume de données important
- Transférer les calculs est moins cher que le transfert de données :
  - un calcul est beaucoup plus efficace s'il est exécuté près des données
  - minimise la congestion du réseau et augmente le débit global du système

### 12.3.1 HDFS

```

1  /* liste les elements */
2  hdfs dfs -ls /user/cloudera/
3  hdfs dfs -mkdir /user/cloudera/dossier
4  hdfs dfs -rm -r /user/cloudera/dossier
5  hdfs dfs -touchz /user/cloudera/file.txt
6  /* transfere de hdfs vers notre home */
7  hdfs dfs -get /user/cloudera/file.txt /home/cloudera/
8  /* transfert de notre home vers notre hdfs */
9  hdfs dfs -put /home/cloudera/file.txt /user/cloudera/

```

Nos fichiers dans Hadoop peuvent être vus dans l'onglet **File Browser**

## 12.4 MapReduce

- Le composant principal du traitement dans l'écosystème Hadoop : il fournit la logique du traitement.
- Des applications qui traitent des données volumineux
- Algorithmes distribués et parallèles

### Paradygme

1. splitting : on découpe les données
2. mapping : on mappe les données pour les utiliser de manière optimisée
3. suffling : on réunit les données selon nos critères
4. reducing : on réduit en effectuant les opérations voulues

**Exemple :**

Etudiant	1	2	3	4	5	6
Département	75	75	75	94	94	69

Comment compter les étudiants dans chaque département ?

script

```

1  > val liste = List((1,75),(2,75),(3,75),(4,94),(5,94),(6,69))
2  > liste.map(x => (1,x._2)).groupBy(_._2).toList.map(x => x._2)
3  res1: List[List[(Int, Int)]] = List(List((1,94), (1,94)), List((1,69)), List((1,75), (1,75), (1,75)))
4  > liste.map(x => (1,x._2)).groupBy(_._2).toList.map(x => x._2).map(x => x.reduce((x,y) => (x._1+y._1,x._2)))
5  res2: List[(Int, Int)] = List((2,94), (1,69), (3,75))

```

## 12.5 Hive

- Créer une base de données relationnelle dans HDFS
- HQL : Hive Query Language = Hive + SQL (SQL Like)
- Facebook a créé Hive pour les personnes habitués à utiliser SQL
- Supporte tous les types primitifs de SQL

### 12.5.1 Lab : Hive CSV SerDe

Uploader les fichiers CSV dans le répertoire hdfs : /user/cloudera/csv/


On utilise le serializer et le deserializer CSV.

Créer la table hive

```
1 CREATE EXTERNAL TABLE etudiant_csv(id string, nom string, age string, departement string)
2 ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
3 WITH SERDEPROPERTIES ("separatorChar" = ",")
4 STORED AS TEXTFILE
5 LOCATION '/user/cloudera/csv/'
6 TBLPROPERTIES ("skip.header.line.count"="1");
```

### 12.5.2 Lab : Hive JSON SerDe

On utilise le serializer et le deserializer JSON.

- Télécharger json-serde 1.3.7 
- Décompresser le contenu
- Copier les jars vers hive : `sudo cp ~/Downloads/jar_files/* /usr/lib/hive/lib/`
- Vérifier la copie des jars : `ls /usr/lib/hive/lib | grep -i json[]`
- Redémarrer le serveur hive : `sudo service hive-server2 restart`
- Uploader les fichiers json dans le répertoire hdfs : /user/cloudera/json/

Créer la table hive

```
1 CREATE EXTERNAL TABLE etudiant_json(id int, nom string, age int, departement string)
2 ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
3 STORED AS TEXTFILE
4 LOCATION '/user/cloudera/json/';
```

En conclusion, Hive crée une base de données SQL virtuelle à partir de fichiers. Chaque opération SQL est gérée par des fonctions MapReduce. De ce fait, une mise à jour des fichiers sera prise en compte dans les requêtes HQL. De même, une requête HQL `INSERT INTO` va générer un nouveau fichier avec nos données. Ce fichier sera par ailleurs lié à une seconde table.

## 12.6 Pig

- Initialement développé par Yahoo!
- Pig Latin : Un langage de haut niveau (un DSL) dédié à l'analyse de gros volumes de données
- Pig runtime : environnement d'exécution (JVM vs Java)
- Il s'adresse aux développeurs habitués à faire des scripts via Bash ou Python par exemple
- Pig est extensible : si une fonction n'est pas disponible, il est possible de l'enrichir via des développements spécifiques dans un langage bas niveau (Java, Python...)
- 10 lignes de Pig Latin = environ 200 lignes de MapReduce Java

### 12.6.1 Lab

- Créer le répertoire `/user/cloudera/drivers-stat/`
- Uploader les fichiers `drivers.csv` et `timesheet.csv` dans ce répertoire

Afficher le contenu de *drivers.csv*

```
1 drivers = LOAD '/user/cloudera/drivers-stat/drivers.csv' USING PigStorage(',');
2 raw_drivers = FILTER drivers BY $0>1;
3 dump raw_drivers;
```

Calculer les heures et les distances effectuées par chaque conducteur

```
1 drivers = LOAD '/user/cloudera/drivers-stat/drivers.csv' USING PigStorage(',');
2 raw_drivers = FILTER drivers BY $0>1;
3 drivers_details = FOREACH raw_drivers GENERATE $0 AS driverId, $1 AS name;
4 timesheet = LOAD '/user/cloudera/drivers-stat/timesheet.csv' USING PigStorage(',');
5 raw_timesheet = FILTER timesheet BY $0>1;
6 timesheet_logged = FOREACH raw_timesheet GENERATE $0 AS driverId, $2 AS hours_logged, $3 AS miles_logged;
7 grp_logged = GROUP timesheet_logged BY driverId;
8 sum_logged = FOREACH grp_logged GENERATE GROUP AS driverId, SUM(timesheet_logged.hours_logged) as
9 sum_hourslogged, SUM(timesheet_logged.miles_logged) AS sum_mileslogged;
10 join_sum_logged = JOIN sum_logged BY driverId, drivers_details BY driverId;
11 join_data = FOREACH join_sum_logged GENERATE $0 AS driverId, $4 AS name, $1 as hours_logged, $2 as
12 miles_logged;
13 dump join_data;
14 store join_data into '/user/cloudera/drivers-stat/result' using PigStorage(',','-schema');
```

## 12.7 Sqoop

- SQL + Hadoop
- Une interface en ligne de commande
- Transférer des données entre Hadoop et SGBDR
- Chargement différentiels d'une table ou d'une requête SQL

### 12.7.1 Lab

```
1 mysql --host=localhost --port=3306 --user=root --password=cloudera
```

```
1 SELECT version(), current_date;
2 SHOW databases;
3 CREATE database test;
4 USE test;
5 CREATE TABLE personne (nom VARCHAR(20), age INT, CONSTRAINT PK_Personne PRIMARY KEY (nom));
6 SHOW tables;
7 DESCRIBE personne;
8 INSERT INTO personne VALUES ('walid', 30);
9 INSERT INTO personne VALUES ('alain', 32);
10 INSERT INTO personne VALUES ('sylvain', 29);
11 SELECT * FROM personne;
```

Import d'une BDD dans HDFS via hive

```
1 sqoop import --connect 'jdbc:mysql://localhost:3306/test?user=root&password=cloudera'
2 --username root --password cloudera --verbose
3 --table personne --hive-import --hive-table personne
```

S'il existe déjà une table du même nom dans Hive, Sqoop va concaténer les nouveaux éléments aux anciens. Pour écraser l'ancienne table, il faut indiquer l'option `--hive-overwrite`. Dans le cas de grosse volumétrie, il est plus judicieux d'utiliser l'option `--query "select * from personne where id > dernier_index"`. Ceci va prendre les éléments à partir d'un certain rang (i.e. le dernier enregistrement).

## Import d'une BDD dans un fichier

```

1 sqoop import --connect 'jdbc:mysql://localhost:3306/test?user=root&password=cloudera'
2   --username root --password cloudera --verbose
3   --table personne --as-textfile

```

Lors d'un import vers un fichier, Sqoop va créer un dossier temporaire selon le nom de la table de la BDD dans le dossier HDFS `cloudera`. Il faut s'assurer qu'il n'existe pas déjà un dossier du même nom pour éviter une erreur d'import. De plus, Sqoop va par défaut répartir les données dans plusieurs fichiers. On peut forcer l'enregistrement dans un seul fichier avec l'option `--num-mappers 1`.

## 12.8 Oozie

- Schedule (Planifie) les jobs Hadoop et les lie comme un travail logique : Oozie Job
  - Oozie prend en charge différents types d'actions : Hadoop MapReduce, les opérations HDFS, Pig, SSH, Sqoop, envoi email...
  - Oozie :
    - notion de workflow : une série d'actions (des jobs Hadoop) formant un graphe orienté acyclique. Le premier noeud est START, le graphe termine par END
    - notion de coordinator : quand lancer un workflow ? Un coordinateur va définir une date de lancement et une périodicité.
- NB : Oozie va vérifier à chaque lancement si les données en entrée sont disponibles. Si elles ne le sont pas, l'exécution est retardée

### 12.8.1 Lab

1. Sauvegarder le script pig 12.6.1 dans un fichier dans HDFS :

```

1 hdfs dfs -put /home/cloudera/Downloads/drivers-stat.pig /user/cloudera/drivers-stat/drivers-stat.pig

```

2. Définir le workflow :

- Pig : créer le lancement d'un script
- Hive : créer une requête pour créer la table des résultats

```

1 CREATE EXTERNAL TABLE timesheet_results (driverId string, nom string, hours_logged string, miles_logged string)
2 ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
3 WITH SERDEPROPERTIES ("separatorChar" = ",")
4 STORED AS TEXTFILE
5 LOCATION '/user/cloudera/drivers-stat/result/';

```

3. Exécuter le workflow
4. Enrichir par deux actions au début du workflow :
  - Hive : suppression de la table s'il existe
  - Script shell : suppression du dossier des résultats
5. Programmer le workflow pour une exécution tout les heures

## 12.9 HBase

- Base de données NoSQL orientée colonnes  
Une base de données orientée objet va gérer comme ceci : 1,alain,2,valid,...
- Un système de stockage par clé/valeur (key/value store)
- Inspiré du projet BigTable de Google
- Écrit en Java

## 12.10 Solr/Lucene

- Moteur de recherche basé sur Lucene
- Open source
- Configuration avec des fichiers xml

## 12.11 TP

### 12.11.1 Oozie avec Pig et Hive

1. Importer les fichiers csv dans HDFS
2. Créer un workflow Oozie pour :
  - avec Pig : calculer le chiffre d'affaires par magasin

```

1 stores = LOAD '/user/cloudera/solar/stores.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage();
2 raw_stores = FILTER stores BY $0>0;
3 stores_details = FOREACH raw_stores GENERATE $0 AS storeId, $1 AS
4     postCode, $2 AS geoLocation, $3 AS name, $4 AS town, $5 AS type;
5 scans = LOAD '/user/cloudera/solar/scans.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage();
6 raw_scans = FILTER scans BY $0>0;
7 scans_logged = FOREACH raw_scans GENERATE $2 AS storeId, $5 AS totalPrice;
8 grp_logged = GROUP scans_logged BY storeId;
9 sum_logged = FOREACH grp_logged GENERATE GROUP AS storeId, SUM(scans_logged.totalPrice) AS
10     sum_totalLogged;
11 join_sum_logged = JOIN sum_logged BY storeId, stores_details BY storeId;
12 join_data = FOREACH join_sum_logged GENERATE $0 AS storeId, $1 AS sum_total, $3 AS
13     postCode, $5 AS name, $6 AS town, $7 AS type, $4 AS geoLocation;
14 DUMP join_data;
15 STORE join_data INTO '/user/cloudera/solar/result' USING PigStorage(',', '-schema');
```

- avec Hive : charger les résultats dans une table

### 12.11.2 Sqoop

1. Importer les fichiers csv (scans.csv et stores.csv) dans MySQL
2. Utiliser Sqoop pour importer une table hive qui calcul le chiffre d'affaires par type de magasin.

Deux tables `scans` et `stores` ont été créées dans la BDD `carrefour`.

Script pour importe un csv dans MySQL

```

1 CREATE TABLE stores (storeId INT NOT NULL AUTO_INCREMENT, product INT,
2     geoLocation VARCHAR(255), name VARCHAR(255),
3     town VARCHAR(255), type VARCHAR(255),
4     PRIMARY KEY (storeId));
5
6 // delete head in your file before
7 LOAD DATA INFILE '/home/cloudera/Downloads/solr/storesmysql.csv'
8     INTO TABLE stores
9     FIELDS TERMINATED BY ','
10    ENCLOSED BY '"'
11    LINES TERMINATED BY '\n';
```

Requête globale SQL

```

1 SELECT scans.storeId, SUM(scans.totalPrice), stores.name FROM scans
2     JOIN stores ON (scans.storeId = stores.storeId) GROUP BY storeId;
```

**Cependant**, Sqoop ne prends pas en charge la fonction `GROUP BY`. Il faut donc importer la table jointe à partir de nos deux tables et utiliser Hive dans la suite pour calculer

cf. § 7.2.3 on <http://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html>

## Import de l'union sélective de nos tables

```
1 sqoop import --connect 'jdbc:mysql://localhost:3306/carrefour?user=root&password=cloudera' \  
2   --username root --password cloudera \  
3   --verbose --query \  
4   'SELECT scans.storeId, scans.totalPrice, stores.name, stores.type FROM scans JOIN stores ON \  
5   (scans.storeId = stores.storeId) WHERE $CONDITIONS' \  
6   --hive-import --hive-table stats \  
7   --target-dir /user/cloudera/solar/statFinal \  
8   --split-by 2
```

La clause WHERE \$CONDITIONS est indispensable pour Sqoop. Cela lui permet de paralléliser l'import. L'option --split-by lui indique en contient de jobs il doit diviser la requête.

## Query Hive

```
1 SELECT stats.type, ROUND(SUM(stats.totalprice)) FROM stats GROUP BY stats.type;
```

L'option ROUND permet de réduire le nombre de décimale des calculs.



# Chapitre 13

## Spark

@Walid Mellouli

### 13.1 Introduction

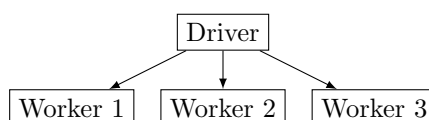
Apache Spark est une alternative à MapReduce d'Hadoop.

#### Spark VS autres

- Framework complet et unifié :
- Nature :
- Type :

Débit RAM		Débit DD	
DDR2	6400 Mo/s	SATA2	250 Mo/s
DDR3	12800 Mo/s	SSD	500 Mo/s

Spark distribue les opérations. On lance notre programme sur un Driver et Spark va distribuer les transformations et/ou actions sur les différents exécuteurs.



#### 13.1.1 Installation

<http://spark.apache.org/downloads.html>

Ajout du path

```
1 export SPARK_HOME=/usr/local/spark
2 export PATH=$PATH:$SPARK_HOME/bin
```

Lancement de Spark

```
1 spark-shell --master local[3]
```

L'option `local` permet d'indiquer le nombre de workers / partition à lancer i.e. le nombre de CPUs pour paralléliser le traitement. Ce paramètre est lié au nombre de noeuds i.e. machines disponibles.

### 13.2 Fonctions



## Premier pas

```

1 > val data =(1 to 1000).toArray
2 // Creation d'une distribution sur data
3 > val distrib = sc.parallelize(data)
4 // affiche le nombre de partition
5 > distrib.partitions.size
6 res1: Int = 3
7 // recupere les donnees dans data
8 > distrib.collect()
9 res2: Array[Int] = Array(1,...,1000)
10 // affiche les donnees de chaque partition
11 > distrib.glom().collect()

```

## Initialisation

```

1 > val data = Array(("A",3),("A",1),("B",5),("A",1),("B",2))
2 > val distData = sc.parallelize(data)

```

## GroupBy

```

1 > distData.groupByKey.collect()
2 res19: Array[(String, Iterable[Int])] = Array((B,CompactBuffer(5, 2)), (A,CompactBuffer(3, 1, 1)))

```

## ReduceByKey

```

1 > distData.reduceByKey((x,y) => x+y).collect()
2 res20: Array[(String, Int)] = Array((B,7), (A,5))

```

## Join

```

1 > val a = sc.parallelize(Array(("A",3),("B",1),("C",5),("D",1),("E",2)))
2 > val b = sc.parallelize(Array(("A","aaa"),("B","b"),("C","cccc"),("D","d"),("E","ee")))
3 > a.join(b).collect()
4 res21: Array[(String, (Int, String))] = Array((B,(1,b)), (E,(2,ee)), (C,(5,cccc)), (A,(3,aaa)), (D,(1,d)))

```

Il existe également les fonctions leftjoin et rightjoin.

## 13.3 DAG : Directed Acyclic Graph

```

1 sc.textFile("/usr/local/spark/README.md").flatMap(_.split(" "))
2   .map(word => (word, 1))
3   .reduceByKey((x,y) => x+y)
4   .collect()

```

<http://spark.apache.org/docs/latest/programming-guide.html#transformations> <http://spark.apache.org/docs/latest/programming-guide.html#actions>

```

1 > sc.textFile("/usr/local/spark/README.md")
2   .flatMap(_.split(" "))
3   .map(word => word.size)
4   .reduce((x,y) => x+y)
5 res8: Int = 3248
6 > sc.textFile("/usr/local/spark/README.md")
7   .flatMap(_.split(" "))
8   .filter(_.contains("tin"))
9   .collect()
10 > sc.parallelize(List(2,5,5,6,2,4,9,8,4,4,8,0,1))
11   .filter(_%2==1).reduce((x,y) => x+y)
12 res16: Int = 20
13 > sc.parallelize(List(2,5,5,6,2,4,9,8,4,4,8,0,1))
14   .filter(_%2==0).map(num => (num,1))
15   .reduceByKey((x,y) => x+y)
16   .collect()
17 res17: Array[(Int, Int)] = Array((0,1), (6,1), (4,3), (8,2), (2,2))

```

### 13.3.1 Création d'un projet avec IntelliJ

1. créer le fichier : 'word-count/src/main/scala/WorldCount.scala'
2. créer un fichier *build.sbt* :

build.sbt

```
1 name := "word-count"
2
3 version := "1.0-SNAPSHOT"
4
5 scalaVersion := "2.11.8"
6
7 libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.1"
```

3. lancer IntelliJ

WorldCount.scala

```
1 import org.apache.spark.rdd.RDD
2 import org.apache.spark.{SparkConf, SparkContext}
3
4 object WordCountDriver extends App {
5   val inputFile = "/usr/local/spark/README.md"
6   val targetDirectory = "word-count-result"
7
8   println("Start Word Count")
9
10  val conf = new SparkConf().setAppName("word-count").setMaster("local[3]")
11  // create a Scala Spark context
12  val sc = new SparkContext(conf)
13  // load our input data
14  val input: RDD[String] = sc.textFile(inputFile)
15  // split up into words
16  val words: RDD[String] = input.flatMap(_.split(" "))
17  // transform into word and count
18  val counts: RDD[(String, Int)] = words.map(word => (word, 1)).reduceByKey((x, y) => x + y)
19  // save the word count back out to a text file, causing evaluation
20  counts.foreach{case (word, count) => println(s"$word: $count")}
21  counts.saveAsTextFile(targetDirectory)
22
23  Thread.sleep(300000)
24  println("Finish !")
25 }
```

## 13.4 TP : Spark RDD

Créer un job spark qui :

- parse le fichier *timesheet.csv*
- calcule pour chaque conducteur la somme des heures loggées et la somme des miles
- log les resultats dans la console
- sauvegarde les résultats dans un fichier text

Commande dans spark-shell

```
1 val test = sc.textFile("/home/inti/10-hadoop/data/pig/timesheet-withoutHead.csv")
2 test.flatMap(_.split("\n")).map(x => x.split(",")).
3   .map(x => (x(0), (x(2).toInt, x(3).toInt)))
4   .reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2))
5   .collect()
```

Script dans IntelliJ

```
1 import java.io.File
2 import org.apache.spark.{SparkConf, SparkContext}
3 import org.apache.spark.rdd.RDD
4
5 object Timesheet extends App {
```

```

6  val inputFile = "src/main/ressources/timesheet.csv"
7  val targetDirectory = "parse-timesheet-result"
8
9  deleteRecursively(new File(targetDirectory))
10
11 println("Start TimeSheet Calculs")
12
13 val conf = new SparkConf().setAppName("parse-timesheet").setMaster("local[3]")
14 // create a Scala Spark context
15 val sc = new SparkContext(conf)
16 val input: RDD[String] = sc.textFile(inputFile)
17 val lines: RDD[String] = input.flatMap(_.split("\n"))
18 val couples :RDD[Array[String]] = lines.map(_.split(","))
19 val couplesFilter :RDD[(String, (Int, Int))] = couples.map(x => (x(0), (x(2).toInt, x(3).toInt)))
20 val counts: RDD[(String, (Int, Int))] = couplesFilter.reduceByKey((x,y) => (x._1+y._1, x._2+y._2))
21 // display results log
22 counts.foreach{case (conducteur, count) => println(s"$conducteur: $count")}
23 counts.saveAsTextFile(targetDirectory)
24 // waiting to close
25 Thread.sleep(300000)
26 println("Finish !")
27 // delete results directory
28 def deleteRecursively(file: File): Unit = {
29     if (file.isDirectory)
30         file.listFiles().foreach(deleteRecursively)
31     if (file.exists && !file.delete)
32         throw new Exception("Cannot delete $(file.getAbsolutePath)")
33 }
34 }

```

## Script dans IntelliJ

```

1  import java.io.File
2  import org.apache.spark.{SparkConf, SparkContext}
3  import org.apache.spark.rdd.RDD
4
5  object Timesheet extends App {
6      case class TimeSheet(driverId: Int, hours: Int, miles: Int)
7      case class Driver(driverId: Int, name: String, location: String)
8      case class DriverTimeSheet(driverId: Int, name: String, location: String, hours: Int, miles: Int)
9
10     val inputFile = "src/main/ressources/timesheet.csv"
11     val inputFileNames = "src/main/ressources/drivers.csv"
12
13     val targetDirectory = "parse-timesheet-result"
14
15     deleteRecursively(new File(targetDirectory))
16
17     println("Start TimeSheet Calculs")
18
19     val conf = new SparkConf().setAppName("parse-timesheet").setMaster("local[*]")
20     // create a Scala Spark context
21     val sc = new SparkContext(conf)
22     val input: RDD[String] = sc.textFile(inputFile)
23     val lines: RDD[TimeSheet] = input.flatMap(_.split("\n"))
24         .map(_.split(","))
25         .map(x => TimeSheet(x(0).toInt, x(2).toInt, x(3).toInt))
26     val counts: RDD[TimeSheet] = lines.map(x => (x.driverId, x))
27         .reduceByKey((x,y) => TimeSheet(x.driverId, x.hours+y.hours, x.miles+y.miles)).map(_._2)
28     val timeSheets: RDD[(Int, TimeSheet)] = counts.map(x => (x.driverId, x))
29
30     val inputNames: RDD[String] = sc.textFile(inputFileNames)
31     val driversNames: RDD[Driver] = inputNames.flatMap(_.split("\n"))
32         .map(x => {
33             val temp = x.split(",")
34             Driver(temp(0).toInt, temp(1), temp(3))
35         })
36     val drivers: RDD[(Int, Driver)] = driversNames.map(x => (x.driverId, x))
37
38     val driverTimeSheet: RDD[DriverTimeSheet] = drivers.join(timeSheets)
39         .map(x => DriverTimeSheet(x._1, x._2._1.name, x._2._1.location, x._2._2.hours, x._2._2.miles))
40         .sortBy(_._2.miles)
41
42     driverTimeSheet.foreach{driver => println {

```

```

43     driver.driverId + "," + driver.name + "," + driver.hours + "," + driver.miles
44   }
45 }
46 driverTimeSheet.saveAsTextFile(targetDirectory)
47
48 Thread.sleep(300000)
49 println("Finish !")
50
51 def deleteRecursively(file: File): Unit = {
52   if (file.isDirectory)
53     file.listFiles().foreach(deleteRecursively)
54   if (file.exists && !file.delete)
55     throw new Exception("Cannot delete ${file.getAbsolutePath}")
56 }
57 }

```

## 13.5 ReduceByKey VS GroupByKey

Le `groupByKey` fait déplacer toutes les données pour les regrouper par groupe avant d'effectuer l'opération voulue. Le `reduceByKey` fait l'opération sur chaque machine et ensuite déplace les données. Étant donné que l'on parle de `reduce`, les données après opération sont diminuées. Le déplacement de données est donc moindre dans le cas du `reduceByKey`.

## 13.6 Cache

Soit une RDD :

```

1 textFile.map().map().foreach()
2 -----.saveAsTextFile()

```

Les `foreach` ou `saveAsTextFile` sont des actions alors que les `map` ou `filter` sont des transformations. Lorsque plusieurs transformations sont utilisées. Lors de la première évaluation, Spark les mets en **cache** pour utiliser le résultats lors d'une autre action appelant ces même transformations. Avec le TP 13.4 précédent, nous avons fait trois actions différentes :

Job Id	Description
0	sortBy
1	foreach
2	saveAsTextFile

## 13.7 Accumulator

En Scala, et de manière générale, dans tout système distribué, on utilise au maximum des variables mutables. Ainsi, on ne peut pas les modifier. Pour permettre un comptage lors d'un parse d'une ressource, on peut alors utiliser un accumulateur pour incrémenter nos opérations.

```

1 > val accum = sc.longAccumulator("Accumulator")
2 > sc.parallelize(Array(1,2,3,4,5)).foreach(x => accum.add(x))
3 > accum.value
4 res87: Long = 15

```

## 13.8 Broadcast

Un **broadcast** s'applique sur une données sous forme de **Map**. Lors de l'utilisation d'une ressource (i.e. fichiers) dans une suite d'opération distribuée, il est nécessaire de la broadcaster afin que chaque exécuter puisse y accéder. En effet, le `map` sert à distribuer les index de notre data sans avoir à envoyer toutes la data à tout le monde. De cette manière, chacun eut demander l'index qui lui faut. Cela permet également la demande simultanée de plusieurs exécuter.

```

1 // transform data to a map
2 > val referential = Array((1, "alain"),(2, "sylvain"),(3, "sophie")).toMap
3 referential: scala.collection.immutable.Map[Int,String] = Map(1 -> alain, 2 -> sylvain, 3 -> sophie)
4 // load a broadcast
5 > val referentialBroadcast = sc.broadcast(referential)
6 > val personnels = sc.parallelize(Array(1,2,3))
7 // recup data
8 > personnels.map(id => (id, referentialBroadcast.value.get(id).get)).collect
9 res88: Array[(Int, String)] = Array((1,alain), (2,sylvain), (3,sophie))

```

## 13.9 Exception Scala

```
1 > val a: Option[Int]
```

a peut avoir la valeur None ou Some(3).

```

1 > val ls :Array[Option[Int]] = Array(Some(3), None, Some(1))
2 > ls.flatten
3 res1: Array[Int] = (3,1)
4 > def tryParse(str: String) :Option[Int] =
5   try {
6     Some(str.toInt)
7   }catch {
8     case e: Exception => None
9   }
10 > tryParse(Array("str",2,5))
11 res2: Array[Int] = (None, Some(2), Some(5))

```

## 13.10 TP 3

But :

- Broadcaster le réf driver en tant que Map(id, driver)
- ajouter des mauvaises ligne et gérer les exceptions
- créer un rapport :
- le nb ligne parsé avec succès
- le nb ligne parsé avec error

Import

```

1 import java.io.File
2 import org.apache.spark.broadcast.Broadcast
3 import org.apache.spark.{SparkConf, SparkContext}
4 import org.apache.spark.rdd.RDD
5 import org.apache.spark.util.LongAccumulator
6
7 case class Report(success: LongAccumulator, error: LongAccumulator)
8 case class Driver(driverId: Int, driverName: String)
9 case class TimeSheet(driverId: Int, hours: Int, miles: Int)
10 case class DriverTimeSheet(driverId: Int, name: String, hours: Int, miles: Int)

```

Main

```

1 object Timesheet extends App{
2
3   import TimeSheetSupport._
4
5   val targetDirectory = "parse-timesheet-result"
6
7   deleteRecursively(new File(targetDirectory))
8   println("Start TimeSheet Calculs")
9
10  val conf = new SparkConf().setAppName("parse-timesheet").setMaster("local[*]")
11  val sc = new SparkContext(conf)

```

```

12
13 val reportTimeSheet = new Report(sc.longAccumulator("Accumulator TimeSheets Success"), sc.longAccumulator("Accumulator TimeSh
14 val reportDriver = new Report(sc.longAccumulator("Accumulator Drivers Success"), sc.longAccumulator("Accumulator Drivers Error
15
16 val drivers: Broadcast[Map[Int, Driver]] = broadcastDrivers(sc)
17 val timeSheets: RDD[(Int, TimeSheet)] = loadTimeSheet(sc, reportTimeSheet)
18
19 timeSheets.reduceByKey((x,y) => TimeSheet(x.driverId,x.hours+y.hours,x.miles+y.miles))
20   .map{ case (driverId, timesheet) =>
21       val driver: Driver = drivers.value.get(driverId).get
22       DriverTimeSheet(driverId, driver.driverName, timesheet.hours, timesheet.miles)
23   }
24   .sortBy(_._2.miles, false)
25   .saveAsTextFile(targetDirectory)
26
27 println(s"Parsing timesheet success: ${reportTimeSheet.success.value}")
28 println(s"Parsing timesheet error: ${reportTimeSheet.error.value}")
29
30 Thread.sleep(300000)
31 println("Finish !")
32 }

```

## Main support

```

1 object TimeSheetSupport {
2   val TimeSheetsFile = "timesheet.csv"
3   val DriversFile = "drivers.csv"
4   def loadTimeSheet(sc: SparkContext, report: Report) = {
5     sc.textFile(getClass.getClassLoader.getResource(TimeSheetsFile).getPath)
6       .flatMap(_._1.split("\n"))
7       .flatMap(line => tryParseTimeSheet(line, report))
8       .map(x => (x.driverId, x))
9   }
10  def tryParseTimeSheet(line: String, report: Report) :Option[TimeSheet] = {
11    val lineTest = line.split(",")
12    try {
13      report.success.add(1)
14      Some(TimeSheet(lineTest(0).toInt, lineTest(2).toInt, lineTest(3).toInt))
15    }catch {
16      case e: Exception => {
17        report.error.add(1)
18        None
19      }
20    }
21  }
22  def broadcastDrivers(sc: SparkContext): Broadcast[Map[Int, Driver]] = {
23    val drivers: Map[Int, Driver] = sc
24      .textFile(getClass.getClassLoader.getResource(DriversFile).getPath)
25      .map(line => parseDriver(line))
26      .map(t => (t.driverId, Driver(t.driverId, t.driverName)))
27      .collect()
28      .toMap
29    sc.broadcast(drivers)
30  }
31  def parseDriver(driver: String): Driver = {
32    val elems: Array[String] = driver.split(",")
33    Driver(elems(0).toInt, elems(1))
34  }
35  def tryParseDriver(line: String, report: Report): Option[Driver] = {
36    val lineTest = line.split(",")
37    try {
38      report.success.add(1)
39      Some(Driver(lineTest(0).toInt, lineTest(1)))
40    }catch {
41      case e: Exception => {
42        report.error.add(1)
43        None
44      }
45    }
46  }
47  def deleteRecursively(file: File): Unit = {
48    if (file.isDirectory)
49      file.listFiles().foreach(deleteRecursively)
50    if (file.exists && !file.delete)

```

```
51     throw new Exception("Cannot delete ${file.getAbsolutePath}")
52   }
53 }
```

# Chapitre 14

# MachineLearning

@Yanik Ngoko

Référence : [4] et [5]

## 14.1 Introduction

Domaine de l'intelligence artificielle :

- Evolutionning calculs
- Planning
- Mutli-system
- Machine learning

Tournant vers les années 70, un statisticien a remplacé un linguiste pour étudier la composition du langage humain dans le but d'établir une grammaire universelle. IBM a ensuite remplacé les linguistes par des statisticiens.

### 14.1.1 Approche

1. classique :  
Problème – > Algorithme -> Solution
2. Machine Learning :  
Problème + Solution – > Algorithme

### 14.1.2 Problèmes fondamentaux

**Le test de Turing** (Imitation Game)

**Problème de l'induction :** Est-il possible d'établir une loi générale à partir d'observations particulières?

### 14.1.3 Apprentissage supervisé

**Régression** Le but consiste à établir une régression logistique ou linéaire pour

**Classification** On établit un réseau entre les entrées et les sorties. La structure a été faite en étudiant la vue et les schémas neuronales.

### 14.1.4 Apprentissage non-supervisé

i.e. clustering ou data-mining



**Association ou Basket-analysis :** études des paniers d'achat dans les magasins pour réorganiser dynamiquement les rayons afin d'associer les produits achetés côte à côte. Pour chaque panier, on analyse le centre commun des paniers et tous les produits qui gravitent autour. Dans le cas non-supervisé, on a seulement les entrées et aucun idée des sorties attendus contrairement à l'apprentissage supervisé.

### 14.1.5 Apprentissage renforcé

cf. alphaGo : logiciel créé par Google ayant gagné face au champion du monde du jeu de Go. Question ouverte : l'AI a-t-elle gagnée parce qu'elle est plus intelligente qu'un Homme ou parce qu'elle a plus de mémoire et une capacité de calcul supérieure ?

### 14.1.6 Induction VS Déduction

**Auguste Comte :** la seule façon de construire la connaissance et de dégager des lois à partir d'observations  
Ex : Newton - Gravitation / Darwin - Théorie de l'évolution Au final, c'est le cheminement de la physique expérimentale. Newton a été corrigé par Einstein, et Newton a corrigé Kepler qui a lui-même corrigé Galilée.

**Karl Popper / David Hume :** limite conceptuelle : régression à l'infini

### 14.1.7 Bilan

En informatique, il y a trois grands problème :

- quand on ne sait rien : on utilise l'IA (Machine Learning)
- quand on sait : on utilise le Génie Informatique ou l'algorithme
- quand on maîtrise : on utilise les maths

## 14.2 Scikit-learn

- <http://scikit-learn.org/>
- projet français, libre
- référence dans le Machine Learning

cf. Tensorflow pour la version américaine créé par Google.

#### Dépendances

```
1 sudo apt-get install build-essential python-dev python-setuptools python-numpy python-scipy libatlas-dev
```

#### Installation

```
1 (pip install numpy)
2 pip install SciPy
3 pip install --user --install-option="--prefix=" -U scikit-learn
4 (ou pip install -U scikit-learn)
```

### 14.2.1 Cloud

cf. doc : <https://computing.qarnot.com/developers/>

- create a Qarnot account and retrieve your personal API token
- need python SDK

### 14.2.2 datasets

- fonction `load_ins()` :
  - attribut `target` : donne la classe de l'élément
  - attribut `etat` : donne l'état de l'élément
- fonction `predict` : pour évaluer notre machine learning sur des tests connus

**limite métrique :** notre machine learning après tests sera juste mais limitée au domaine qu'on lui a donnée d'étudier.

#### Deux scripts possible

- Iris.py : script de traitement
- Qarnotscript.py : script de déploiement sur le cloud

### 14.2.3 Réponses de l'apprentissage supervisé

1. Features :
  - vecteurs de réels (i.e. hauteur, poids) pour des réponses directes
  - classes (i.e. reptiles, mammifères ...) pour des réponses numérisées
2. Modèle Baysésien, SVM, réseaux de convolution ...
3. Problème de l'overfitting : validation croisée, learning curves, scores ...  
Il ne faut pas vouloir passer par tous les points de nos données mais il faut trouver le juste milieu pour ne pas avoir un modèle trop simpliste.

**Attention** il ne faut pas faire complètement confiance en la distribution de nos données.

La solution est de prendre des sous-ensembles de nos données pour construire notre modèle et de le tester le reste des données. La croisée de nos résultats va permettre d'obtenir le modèle le plus juste.

### 14.2.4 Pipelines standards

1. À partir de données brutes, on crée des **features** i.e. extracteur de données
2. Construction du modèle
- 3.

### 14.2.5 Rasoir d'Occam

Lors de la construction d'un modèle, si une colonne n'apporte pas d'information mais uniquement du bruit il n'est pas nécessaire de la conserver.

## 14.3 Features

### 14.3.1 Observations série-temporelles

Ce sont des séries bi-dimensionnelles. La 1<sup>ère</sup> renvoie une classe d'observation et la 2<sup>de</sup> le temps. Il faut trouver le bon découpage temporel.

### 14.3.2 MFCCs : traitement du son

Outils d'extraction à partir de wav :

- YaFee
- PyaudioAnalysis

### 14.3.3 Haar features : reconnaissance des images

Outil : OpenCV Le principe est basé sur les niveaux de contraste.

### 14.3.4 Bag of words : reconnaissance de texte

Librairie python : NLTK permet de faire la tokenisation d'un texte.

## 14.4 Construction de modèles

i.e. classificateurs

#### Modèle SVM

```
1 from sklearn.svm import SVC
2 ...
3 model = SVC()
```

#### Out

```
1 SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
2     decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
3     max_iter=-1, probability=False, random_state=None, shrinking=True,
4     tol=0.001, verbose=False)
```

#### Modèle Baysésien

```
1 from sklearn.tree import DecisionTreeClassifier
2 ...
3 model = DecisionTreeClassifier()
```

#### Out

```
1 DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
2                        max_features=None, max_leaf_nodes=None,
3                        min_impurity_split=1e-07, min_samples_leaf=1,
4                        min_samples_split=2, min_weight_fraction_leaf=0.0,
5                        presort=False, random_state=None, splitter='best')
```

#### Modèle Gaussien

```
1 from sklearn.naive_bayes import GaussianNB
2 ...
3 model = GaussianNB()
```

#### Out

```
1 GaussianNB(priors=None)
```

KNN : méthodes des plus proches voisins. On se fixe une limite de résultats les plus proches de notre recherche.

#### Méthodes :

1. privilégier la construction parallèle de plusieurs modèles à la fois.
2. tester toutes les combinaisons possibles des hyper-paramètres des modèles.
3. sous-échantillonner les données

## 14.5 Metrics

Vrai - positif	Faux - positif
Faux - négatif	Vrai - négatif

La précision se lit horizontalement.

$$precision = \frac{True+}{True++False+} \quad (14.1)$$

Quand le modèle prédit un résultat vrai et que la prédiction est bonne. Le modèle est précis.

La sensibilité (`recall`) se lit verticalement.

$$recall = \frac{\text{ff}}{\text{ff}} \quad (14.2)$$

$$F1 = 2 \times \frac{precision * recall}{precision + recall} \quad (14.3)$$

## 14.6 Learning Curves

## 14.7 TP Fleurs

Le but est de reconnaître trois type d'Iris. Dans un premier temps, il faut absolument la connaissance métier. Dans notre cas, on va comparer des iris selon la longueur et largeur des pétales et sépales. Il faut donc s'assure que ces caractéristiques sont pertinentes pour distinguer les types d'iris.

script

```

1 from sklearn import datasets
2 from sklearn import metrics
3 from sklearn.tree import DecisionTreeClassifier
4 # load the iris datasets
5 dataset = datasets.load_iris()
6 #fit a CART model to the data
7 model = DecisionTreeClassifier()
8 model.fit(dataset.data, dataset.target)
9 # make predictions
10 expected = dataset.target
11 predicted = model.predict(dataset.data)
12 # summarize the fit of the model
13 print(metrics.classification_report(expected, predicted))

```

## 14.8 Apprentissage non-supervisé

Deux principes :

1. non-hiérarchique (algo de Lloyd) : on fait directement les  $k$  clusters voulus et on les remplit (cf. `KMeans`).
2. hiérarchique : on regroupe progressivement les données en cluster (cf. `AgglomerativeCluster`).

On considère un ensemble fini de features  $X = x_1, x_2, \dots, x_n$ . Dans le cas des iris, peux-t-on retrouver les trois types d'iris que l'on étudie ?

Modèle KMeans

```

1 from sklearn.cluster import KMeans
2 model = KMeans(n_clusters = 3)
3 model.fit(dataset.data)

```

Out

```

1 KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
2         n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
3         random_state=None, tol=0.0001, verbose=0)

```

Par défaut, `KMeans` calcul la distance euclidienne entre chaque donnée. Il partitionne ensuite les données en clusters de façon à équilibrer les données. (Autres distances : Cambera, Minkowsky, Manhattan). Il établit ainsi un nombre de centre fixé ayant en moyenne une même distance avec les points voisins. Toutes nos données sont donc reliées à un centre.

Cependant, certaines dimensions des features peuvent être liées et entraînent des problèmes de corrélation. De fait, le calcul de la distance n'est pas appropriée pour clusteriser. Il existe différents modèles pour calculer le coefficient de corrélation entre les données :

#### Modèle AgglomerativeClustering

```
1 from sklearn.cluster import AgglomerativeClustering
2 model = AgglomerativeClustering(3, affinity='euclidean')
3 model.fit_predict(dataset.data)
```

#### Out

```
1 AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
2     connectivity=None, linkage='average',
3     memory=Memory(cachedir=None), n_clusters=3,
4     pooling_func=<function mean at 0x7ff49813fd90>)
```

On partitionne les données. A chaque itération, on regroupe les données jugées proches jusqu'à obtenir le nombre de catégories voulues.

## 14.9 TP Tweets

cf. <http://scikit-learn.org/examples> L'établissement d'un modèle de Machine Learning s'effectue sur des calculs de données. Dans notre cas de Tweets sous forme de texte, il faut donc transposer les tweets texte en nombre. Pour ce faire, il existe plusieurs méthodes d'analyse de texte : Bag of words, Tfidf, ...

#### Lecture des données d'entrée

```
1 def loadTxt(path):
2     return open(path).read().replace("\n", "")
3 data = []
4 target = []
5 for file in os.popen("ls txt_sentoken/pos").readlines():
6     data.append(loadTxt("txt_sentoken/pos/"+file.strip()))
7     target.append(1)
8 for file in os.popen("ls txt_sentoken/neg").readlines():
9     data.append(loadTxt("txt_sentoken/neg/"+file.strip()))
10    target.append(0)
```

`data` représente une liste de tweet positif puis négatif. Cette liste est mise en correspondance avec `target` constitué uniquement de 0 et de 1 (1 = positif, 0 = négatif).

### 14.9.1 Bag of words : CountVectorizer

Cette méthode compte le nombre d'occurrence de chaque mot. L'importance des mots est représenté par leur nombre d'occurrence. Toutefois, il est possible d'indiquer un seuil minimal avec le paramètre `min_df` (représente la fréquence d'apparition minimale d'un mot pour le juger important).

```
1 > tf_vectorizer = CountVectorizer(min_df=1)
2 > data_samples = ["we are on monday", "we are on tuesday"]
3 > tf = tf_vectorizer.fit_transform(data_samples)
4 > print(tf_vectorizer.vocabulary_)
5 {'we': 4, 'on': 2, 'are': 0, 'monday': 1, 'tuesday': 3}
6 > print(tf)
7 (0, 1) 1
8 (0, 2) 1
9 (0, 0) 1
10 (0, 4) 1
11 (1, 3) 1
12 (1, 2) 1
13 (1, 0) 1
14 (1, 4) 1
```

```

15 > print(tf.toarray())
16 [[11101]
17  [10111]]

```

$$tf(t, d) = \frac{f(t, d)}{\sum_{t' \in d} f(t', d)} \quad (14.4)$$

Définition

```

1 from sklearn.feature_extraction.text import CountVectorizer
2 def countVectorizerFit(data_samples, pMinDf=1):
3     tf_vectorizer = CountVectorizer(min_df=pMinDf)
4     tf = tf_vectorizer.fit_transform(data_samples)
5     return tf
6 dataFit = countVectorizerFit(data)

```

Une fois cette transformation faite, il est possible de construire un modèle sur ces données d'entrée.

### 14.9.2 Tfidf : TfidfVectorizer

Cette compte le taux d'apparition de chaque mot en pondérant leur nombre d'occurrence par rapport aux occurrences de tous les mots trouvés. Ce mode est le plus pertinent.

$$idf(t, d) = \log \frac{N}{|\{d \in D : t \in d\}|} \quad (14.5)$$

$$tfidf(t, d) = tf(t, d).idf(t, d) \quad (14.6)$$

Le terme  $\{d \in D : t \in d\}$  peut être assimilé à l'entropie de chaque mot dans le document.

Définition

```

1 from sklearn.feature_extraction.text import TfidfVectorizer
2 def tfidfVectorizerFit(data_samples, pMinDf=1):
3     tfidf_vectorizer = TfidfVectorizer(min_df=pMinDf)
4     tfidf = tfidf_vectorizer.fit_transform(data_samples)
5     return tfidf
6 dataFit = tfidfVectorizerFit(data)

```

### 14.9.3 Méthodologie

Soit une base de données, une méthode est d'établir un modèle en se basant sur la moitié des données et on teste notre modèle sur l'autre moitié.

Par exemple

```

1 # une moitié pour le modele
2 train = dataFit[0::2]
3 trainTarget = target[0::2]
4 # une moitié pour le test
5 test = dataFit[1::2]
6 testTarget = target[1::2]

```

Rappel : `target` représente l'état possible des textes : soit positif (1) soit négatif (0).

### 14.9.4 Validation croisée

La validation croisée consiste à créer sur une partie des données une subdivision de données. On construit alors un modèle en ôtant une division et on teste sur cette division ôtée. Cette opération peut-être automatisée à l'aide de plusieurs méthodes : Leave-one-out et K-fold.

cf. scikit-learn-cross-validation méthode bayes multinomial les plus pertinentes dans l'état de l'art de machine learning

## Leave One Out

```

1  from sklearn.model_selection import LeaveOneOut
2  from sklearn.naive_bayes import GaussianNB
3
4  loo = LeaveOneOut()
5  X = train
6  f1 = []
7  for train_index, test_index in loo.split(X):
8      X_train = train[train_index]
9      X_test = train[test_index]
10     X_target = []
11     for index in train_index:
12         X_target.append(trainTarget[index])
13     X_targetTest = []
14     for index in test_index:
15         X_targetTest.append(trainTarget[index])
16     model = GaussianNB()
17     model.fit(X_train.toarray(), X_target)
18     predicted = model.predict(X_test.toarray())
19     expected = X_targetTest
20     print(metrics.f1_score(expected, predicted, average='micro'))
21 print(sum(f1)/len(f1))

```

## K-Fold

```

1  from sklearn import datasets
2  from sklearn import metrics
3  from sklearn.tree import DecisionTreeClassifier
4  from sklearn import cross_validation
5  dataset = datasets.load_iris()
6  def clfTree(data, target):
7      model = DecisionTreeClassifier()
8      model.fit(data, target)
9      return model
10 def testModel(model, data, target):
11     expected = target
12     try:
13         predicted = model.predict(data)
14     except Exception:
15         predicted = model.fit_predict(data)
16     print(metrics.classification_report(expected, predicted))
17     print('-----\n')
18     return metrics.f1_score(expected, predicted, average='micro')
19 # create 4 folds
20 kf_total = cross_validation.KFold(len(dataset.data), n_folds=4, shuffle=True, random_state=4)
21 for train_index, test_index in kf_total:
22     testModel(clfTree(x[train_index], y[train_index]), x[test_index], y[test_index])

```

# Chapitre 15

## Visualisation

@Yanik Ngoko

Quelques outils :

- Matplotlib
- Gnuplot
- D3js
- Spark

### 15.1 Spark Notebook

cf. Scala Data Scientist

Télécharger `spark-notebook.deb` sur le site `spark-notebook.io` puis le projet Git.

Exemple

```
1 case class Species(name: String, amount: Int, poids: Int, zoo: String)
2 val zoos = Seq(Species("giraffes", 8,350,"SF Zoo"),
3                 Species("orangutans", 12,80,"SF Zoo"),
4                 Species("monkeys", 18,45,"SF Zoo"),
5                 Species("canard", 23,2,"SF Zoo"),
6                 Species("giraffes", 6,350,"LA Zoo"),
7                 Species("orangutans", 14,80,"LA Zoo"),
8                 Species("monkeys", 15,45,"LA Zoo"),
9                 Species("canard", 28,2,"LA Zoo"),
10                Species("giraffes", 3,350,"NY zoo"),
11                Species("orangutans", 18,80,"NY zoo"),
12                Species("monkeys", 12,45,"NY zoo"),
13                Species("canard", 0,2,"NY zoo"),
14                Species("giraffes", 4,350,"AZ zoo"),
15                Species("orangutans", 12,80,"AZ zoo"),
16                Species("monkeys", 20,45,"AZ zoo"),
17                Species("canard", 5,2,"AZ zoo"))
18 // affiche en barre selon la quantite
19 CustomPlotlyChart(zoos,
20                   dataOptions="{type: 'bar'}",
21                   dataSources="{x: 'name', y: 'amount'}")
22 // group par zoo et affichage en barre
23 CustomPlotlyChart(zoos, layout="{barmode: 'group'}",
24                   dataOptions="{splitBy: 'zoo', type: 'bar'}",
25                   dataSources="{x: 'name', y: 'amount'}")
26 // group par zoo et affichage en empilement
27 CustomPlotlyChart(zoos, layout="{barmode: 'stack'}",
28                   dataOptions="{splitBy: 'zoo', type: 'bar'}",
29                   dataSources="{x: 'name', y: 'amount'}")
```

L'ensemble des meta-class prédéfinies sont visibles dans le dossier :  
`spark-notebook-master/modules/common/src/main/scala/notebook/front/widgets/charts.`



### 15.1.1 TP CO2

Récupérer la base de données des émissions de CO2 sur le site [data.gouv.fr](https://data.gouv.fr)

Chargement de la BDD

```
1 case class Cars(marque: String, model: String, co2: Float)
2 val sqlContext = new SQLContext(sc)
3 val df = sqlContext.read
4   .format("com.databricks.spark.csv")
5   .option("header", "true")
6   .option("inferSchema", "true")
7   .option("delimiter", ";")
8   .load("mars-2014-full.csv")
```

Tri sur les colonnes

```
1 val renault = df.select("lib_mrqr", "lib_mod_doss", "co2", "conso_mixte").filter("lib_mrqr = 'RENAULT'")
2 val toyota = df.select("lib_mrqr", "lib_mod_doss", "co2", "conso_mixte").filter("lib_mrqr = 'TOYOTA'")
```

Graphiques

```
1 CustomPlotlyChart(toyota, layout="{barmode: 'group'}",
2   dataOptions="{splitBy: 'lib_mod_doss', type: 'bar'}", dataSources="{x: 'lib_mrqr', y: 'conso_mixte'}")
3 CustomPlotlyChart(renault, layout="{barmode: 'group'}",
4   dataOptions="{splitBy: 'lib_mod_doss', type: 'bar'}", dataSources="{x: 'lib_mod_doss', y: 'co2'}")
```

Calcul de moyenne

```
1 val test = df.select("lib_mrqr", "lib_mod_doss", "co2")
2 val statAvgByModel = test.groupBy(test("lib_mrqr"), test("lib_mod_doss")).agg(avg(test("co2")) as "avg-co2")
3 val statAvgByMarque = test.groupBy(test("lib_mrqr")).agg(avg(test("co2")) as "avg-co2")
```

Graphique

```
1 CustomPlotlyChart(statAvgByMarque, layout="{barmode: 'group'}",
2   dataOptions="{splitBy: 'lib_mrqr', type: 'bar'}", dataSources="{x: 'lib_mrqr', y: 'avg-co2'}", )
```

### 15.1.2 Diagramme

Téléchargement de données

```
1 val geoJsonsDir = "https://api.github.com/repos/poezn/us-history-maps/contents/GeoJSON?ref=master"
2 val listingString = scala.io.Source.fromURL(geoJsonsDir).getLines.mkString
3 val listing = play.api.libs.json.Json.parse(listingString)
```

String to Array

```
1 > val play.api.libs.json.JsonArray(arr) = listing
2 arr: Seq[play.api.libs.json.JsValue]
```

Tri et map

```
1 > val geoJsons = arr.map(x => (x \ "name").as[String])
2   .filterNot(_ == ".DS_Store")
3   .filterNot(_.startsWith("mapgroup"))
4   .filter(_.nonEmpty)
5   .map(x => (x.takeWhile(_.isDigit).toInt, s"https://raw.githubusercontent.com/poezn/us-history-maps/master/GeoJSON/$x"))
6   .sortBy(_._1)
7 geoJsons: Seq[(Int, String)]
```

Visualisation

```
1 val minDate = geoJsons.sorted.head._1
2 LineChart(geoJsons.map(_._1) zip geoJsons.map(_._1 - minDate).toList.sorted)
3 // BarChart(geoJsons.map(_._1) zip geoJsons.map(_._1 - minDate).toList.sorted)
```

## 15.2 Kibana

cf. 20.2

# Chapitre 16

## Base de données NoSQL

@Olivier Curé

- Maître de conférence LIGM : site
- Domaine : Big data, Artificial Intelligence, Semantic Web :  
article en 2001 - notion d'ontologie. On réfléchit en inférence et déduction. Google commence à l'incorporer en 2013. Le Web actuel est syntaxique alors que le Web sémantique est conceptuel.

### 16.1 Histoire

- a) Flat model (50s) :  
premier langage de programmation par les scientifiques : Fortran calcul de haute précision.  
deuxième langage par la finance : Coboll (peur au passage à l'an 2000, l'encodage des dates étaient fait sur les deux derniers chiffres de l'année)  
Problems :
  - Data redundancy
  - At update-time
  - No abstract data model
  - Requires some knowledge on the physical storage organization
  - No common query language
- b) Hierarchical model (60s) : Tree-like model  
Single upward link in each record  
Record are sorted  
Problems :
  - Modifying data structures imposes to change programs
  - Programmers need some time to get used to the database structure.  
Systems : North American Rockwell & IBM : IMS  
(Information Management System)
- c) Network model (70s) : An extension of the hierarchical model  
Standardisation in 1971 by the CODASYL group (Conference on Data Systems languages)  
Two fundamental constructs : records and sets.  
Records contain fields  
Sets define one-to-many relationships between records : one owner, many members.  
Problems : navigation in the DB is harder than in hierarchical model.  
Systems : IDMS (Integrated Database Management System)
- d) Relational model (80s) : Proposed by E.F. Codd (IBM) in early 70s.  
Relations are connected (primary key, foreign key)  
Declarative query language : SQL -> optimization  
- Set based operations (relational algebra)  
1<sup>st</sup> Systems : System R, Ingres (1971, UC Berkeley with QUEL)-> Postgres (1985), Oracle (1977 with

SQL), DB2 (IBM, 1984 with SQL)

chercheur chez IBM qui présente son idée. Un type est intéressé et monte une start-up : Oracle.

SQL Server : Microsoft

DB2 : IBM

MySQL : Oracle

e) Object oriented model (90s) :

scala, java, c++ : on lie des données dans des bases de données et on les mets dans des objets.

Les grands des BDD relationnels vont s'adapter à l'orienté objet et donc vont tuer les nouveaux arrivants.

f) Semi-structured, XML (2000s) :

On pensait que XML allait manger le HTTL. Idée de créer des bases de données XML. Anéanti par les grands qui ont intégré des méthodes pour gérer les commandes XML.

- Relational : OLTP (OnLine Transactional Protocol) on enregistre tout et le plus vite possible (Record chez NetFlix avec 1 million de transaction /sec). Tout est ensuite analysé pour augmenter le chiffre d'affaire. On fait beaucoup de petites transactions à la seconde (ex : vente de billets de train).
- embedded : base de données embarquée
- memory centric : enregistrement dans la RAM. + rapide mais pas de persistance. Nécessité de répliquer sur plusieurs data center.
- data warehouse : OLAP (OnLine Analytical Protocol) tout est analysé. On fait de grosse transaction mais à une faible fréquence (ex : analyse l'organisation des rayons d'un supermarché).

### 16.1.1 Two forms of Big volumes

- 'small analytics' : SQL on very large data sets. Using aggregate ops of SQL.
- 'big analytics' : Data clustering, regressions, machine learning. Using statistical tools : R, SPSS (Statistical Product and Service Solutions - IBM), SAS.

### 16.1.2 Making sense at scale

- Machines : cloud computing
- Algorithms : machine learning and analytics (Cours d'Etienne Côme)
- People : crowdsourcing and human computation
- Edge computing : distribué les calculs sur les data center plutôt que d'envoyer les données et saturer le réseau.

### 16.1.3 Crowdsourcing and Human computation

**Crowdsourcing** first coined in 2006 in Wired magazine : 'a task of taking a job traditionally performed by a designated agent and outsourcing it to an undefined, generally large group of people in the form of an open call'.

**Human computation** (van Hahn , 05) : '... a paradigm for utilizing human processing power to solve problems that computers cannot yet solve'.

**Global Brain** (Bernstein, CACM15) : people and computers to constitute a global brain. Ask for new programming metaphors to program it.

## 16.2 NoSQL : Not only SQL

Organisation dans la Silicon Valley d'un meet-up pour parler d'une idée de base de données non SQL. Les grands acteurs sont Google et Amazon. Ceux sont les premiers à être confronté à des collectes de données en grand volume et de structure différente.

- permette la montée en charge horizontalement tout en gérant la montée des données et sans augmenter les temps de traitement
- temps de latence réduit sur des requêtes régulière.

### 16.2.1 Motivation

Les grandes sociétés sont arrivés au point de devoir distribuer leurs données. Le problème de la distribution des bases données engendrent la parallélisation des requêtes. Les systèmes de données relationnels ne sont pas prévus et optimisé pour cette pratique. Le NoSQL est né de la nécessité de requête en parallèle. Deuzio, il n'y a plus de schéma et donc la BDD accepte tous types de document. Tertio, il faut permettre le scale-out .i.e. l'augmentation de machines et de données tout en gardant les performances.

- (a) RDBMS have been successful for more than twenty years, providing standards (SQL), persistence, concurrency control, and an integration mechanism (several apps accessing a single database).  
Le mécanisme d'intégration
- (b) App developers have been frustrated with the impedance mismatch between the relational model and the in-memory data structures.  
Pas d'optimisation des requêtes comme SQL. SQL est déclarative : le système réécrit notre requête de façon optimisé avec ses algorithmes. NoSQL est procédurale : l'utilisateur doit programmer comment le système doit gérer la requête.
- (c) Exponential growth of data set size (161Eo (2006) to 988Eo (2010) created and replicated data) requires a cluster-based approach to DBMS.
- (d) RDBMS are not designed to run efficiently on a cluster (e.g. Oracle RAC and SQL Server clustered version use a shared-disk approach -> disk system as a SPOF : Single Point Of Failure). MongoDB est en SPOF, Cassandra ne l'est pas.

**Definition** (Forrester Research Inc.) "a no-relational database management system that provides storage, processing, and accessing of complex data structures and support for large volumes of polystuctured data. It supports a horizontal, on-demand, extreme scale-out database platform tat delivers a schema-less and flexible data model, and is optimizd for high performance."

### 16.2.2 Catégories

1. Key-value stores : association d'une valeur avec une clé  
Pour chaque clé, on a une donnée.
  - Dynamo (Amazon)
  - Voldemort (LinkedIn)
2. Document DB : association d'un document JSON avec une clé  
Pour chaque clé, on a un JSON.
  - MongoDB (Open Source)
  - Terrastore
3. Column family DB : association d'une famille de clé-valeur avec une clé  
Pour chaque clé, on donne une structure de données sous forme de clé/valeur/timestamp.
  - BigTable (Google)
  - Cassandra (Facebook puis Apache)
  - HBase (Apache)
4. Graph DB : inter-connexion entre plusieurs données.
  - Neo4j

### 16.2.3 ACID

**Atomicité** : Tout où rien. Sur un ensemble d'opération, soit tout est réalisé correctement soit tout est annulé.

**Consistence :** Garantie que l'on est toujours dans un état cohérent.

**Isolation :** Chaque requête est isolée. Un **SELECT** ne verra pas les nouvelles insertions d'un **INSERT** tant que celui n'aura pas fait de **COMMIT**.

**Durabilité :** Persistance des données.

## 16.3 TP1

cf. 2.2.3

## 16.4 MongoDB

**Une base de donnée orienté document :** Un document est un tableau associatif typé :

- PHP Array
- Ruby Hash
- Python Dict
- Objet JSON

**Haute performance :**

- écrit en C++
- données sérialisées en BSON (version compressé de JSON)
- index primaire et secondaire (accès rapide au document et à l'intérieur du document)
- modèle document impose moins de tâche I/O
- moteur de stockage
  - Memory-mapped files
  - WiredTiger

**Fonctionnalité :**

- Langage de requête
- Agrégation
- Géospatial
- Support pour de nombreux langages
- Flexibilité des schémas i.e. il n'y a pas de champ NULL
- Réplication

	SGBDR	MongoDB
<b>Correspondance :</b>	Table, view	Collection
	Row	Document
	Jointure	Document imbriqué
	Clé étrangère	Reference
	Partionnement	Sharp

**Compromis :**

- Pas de jointure
- Pas de transaction ACID
- Pas de schéma
- Pas de durabilité sur une seule machine

**Modélisation :** en orienté document il faut tenir compte

- Du rapport entre lecture et écriture sur les éléments de la base
- Taux de modification des éléments sur lesquels les modifications sont effectuées.
- Taux de croissance du document

### 16.4.1 Prise en main

#### Installation

```
1 /* Installation */
2 sudo apt-get install mongodb
3 /* Lancement du server */
4 mongod --dbpath /home/inti/data/mongodb/
```

#### Lancement de mongo en console

```
1 mongo
```

On écrit les requêtes dans le shell de MongoDB sous forme de scripts Javascript :

- liste des DB : > `show dbs`
- utiliser une BD : > `use dbName`
- liste des collections d'une BD : > `show collections`
- Utilisation de l'approche orientée objet de JS (opérande `.`) : > `db.nomCollection.operation(...)`

#### Création d'un document

```
1 // create a JSON
2 > md = {"nom":"Davis","prenom":"Miles","adresse":"NY, USA"}
3 // Use new database or create if not exist
4 > use person
5 // insert JSON in collection "jazzmen" on database "person"
6 > db.jazzmen.insert(md)
```

- récupération d'un document : > `db.collection.find()`
- affichage bien formater : `.pretty()`
- filtrer les données :

```
1 > db.collection.find({attribut:valeur, ...})
```

- filtrer les données et afficher uniquement les éléments voulus :

```
1 > db.collection.find({attribut:valeur}, {element1:valeur1},...)
```

valeur1 = 0 (false) ou 1 (true)

### 16.4.2 Insertion

- Insertion d'un unique document :

```
1 > db.maCollection.insertOne({ .. })
```

- Insertion d'un ensemble de documents :

```
1 > db.maCollection.insertMany([d1,d2])
2 > db.maCollection.insert({ .. })
```

- On peut aussi mettre une variable JS comme paramètre

### 16.4.3 Sélecteurs

#### Comparaison

- \$eq Matches values that are equal to a specified value.
- \$gt Matches values that are greater than a specified value.
- \$gte Matches values that are greater than or equal to a specified value.
- \$lt Matches values that are less than a specified value.
- \$lte Matches values that are less than or equal to a specified value.
- \$ne Matches all values that are not equal to a specified value.
- \$in Matches any of the values specified in an array.
- \$nin Matches none of the values specified in an array.

#### Exemple

```
1 > db.collection.find({annee : {$eq : "2017"}})
2 > db.collection.find({annee : {$in : ["2016", "2017"]}})
```

#### Logique

- \$or Joins query clauses with a logical OR returns all documents that match the conditions of either clause.
- \$and Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
- \$not Inverts the effect of a query expression and returns documents that do not match the query expression.
- \$nor Joins query clauses with a logical NOR returns all documents that fail to match both clauses.

#### Exemple

```
1 > db.collection.find({annee : {$or : [{gender : "Male"}, {age : "27"} ]}})
2 > db.collection.find({annee : {$not : {$lte : "2016"}}})
```

#### Element

- \$exists Matches documents that have the specified field.
- \$type Selects documents if a field is of the specified type.

#### Exemple

```
1 > db.maCollection.find({annee : {$exists : 1}})
2 > db.maCollection.find({annee : {$type : 2}})
```

Type 2 est String. cf. docs.mongodb.com pour une liste des types.

#### Evaluation

- \$mod Performs a modulo operation on the value of a field and selects documents with a specified result.
- \$regex Selects documents where values match a specified regular expression.
- \$text Performs text search. (nécessite un index)
- \$where Matches documents that satisfy a JavaScript expression.

#### Exemple

```
1 > db.maCollection.find({annee : {$mod : [divisor, remainder]}})
2   {<field>: {$regex: /pattern/, $options: '<options>' }}
3   {<field>: {$regex: 'pattern', $options: '<options>' }}
4   {<field>: {$regex: /pattern/<options> }}
5 > db.movies.find({title : {$regex: 'iolence'}},{title:1, year:1})
```

Sélecteurs : \$text

```

1 > db.articles.createIndex( {subject: "text" })
2 > db.articles.find( {$text: {$search: "coffee" }})

```

### Sélecteurs sur les tableaux

\$all Matches arrays that contain all elements specified in the query.

\$elemMatch Selects documents if element in the array field matches all the specified \$elemMatch conditions.

\$size Selects documents if the array field is a specified size.

Exemple

```

1 > db.maCollection.find({drinks :{$all :["Coffee","Tea"]}})
2 > db.maCollection.find({drinks :{$size :2}})
3 > db.maCollection.find({personnes :{$elemMatch :{prenom : "Marie"}}})

```

### 16.4.4 Projections

\$ Projects the first element in an array that matches the query condition.

\$elemMatchProjects the first element in an array that matches the specified \$elemMatch condition.

\$meta Projects the document's score assigned during \$text operation.

\$slice Limits the number of elements projected from an array. Supports skip and limit slices.

Exemple

```

1 > db.maCollection.find({annee :{$eq :2005}},{"personnes.$":1})
2 > db.maCollection.find({annee :{$eq :2005}},{personnes :{$slice:2}})

```

## 16.5 TP2

Tri sur les années

```

1 > db.movie.find({year :{$gte:"2005"}}).pretty()
2 // display title only
3 > db.movie.find({year :{$gte:"2005"}},{title:1}).pretty()
4 // display title only without id
5 > db.movie.find({year :{$gte:"2005"}},{title:1, _id:0}).pretty()
6 // display all without title
7 > db.movie.find({year :{$gte:"2005"}},{title:0}).pretty()

```

Tri sur les titres

```

1 // display actors of a movie
2 > db.movie.find({title :{$regex: 'movieName'}},{actors:1, title:1, _id:0}).pretty()
3 // display summary of a movie
4 > db.movie.find({title :{$regex: 'movieName'}},{summary:1, title:1, _id:0}).pretty()
5 // display director
6 > db.movie.find({title :{$regex: 'movieName'}},{director:1, title:1, _id:0}).pretty()

```

Tri sur les réalisateurs

```

1 // display movies with director firstname is David
2 > db.movie.find({"director.firstname":"David"},{title:1, _id:0}).pretty()
3 // display director lastname with director firstname is David
4 > db.movie.find({"director.firstname":"David"},{"director.lastname":1, _id:0}).pretty()
5 // display movies with actor firstname is Vigo
6 > db.movie.find({"actors": {$elemMatch :{"firstname":"Vigo"}}},{title:1, _id:0}).pretty()
7 // display director lastname with birthdate is before 1960
8 > db.movie.find({"director.birthdate": {$gte : "1960"}},{director.lastname:1, _id:0}).pretty()

```



Test logique

```
1 // test if exist field "country"
2 > db.movie.find({"country": {$exists:1}}, {title:1, _id:0}).pretty()
3 // display movie US or drama
4 > db.movie.find({$or: [{country:"USA"}, {"genre":"Drama"}]}, {title:1, _id:0}).pretty()
```

Mettre à jour un document en entier

```
1 > var into = db.movie.findOne({title:"Into the Wild"})
2 > into.summary = "After grad..."
3 > into.actors = []
4 > db.movie.update({_id :ObjectId("592d6d200b56f5ae8fc07a8e")}, into)
```

Mettre à jour un élément d'un document

```
1 // initialise la liste des acteurs
2 > db.movie.findOneAndUpdate({title:"Into the Wild"},
3   {$set :{"actors" :[{ "lastname":"Hirsch", "firstname":"Emile", "birthdate":"1985", "role":"Chirs McCandless"}]}})
4 // ajoute un acteur
5 > db.movie.findOneAndUpdate({title:"Into the Wild"},
6   {$addToSet :{"actors" :[{ "lastname":"Hurt", "firstname":"William", "birthdate":"1950", "role":"Walt McCandless"}]}})
```

Log pour assurer l'acidité Lock pour assurer l'isolation

## 16.6 Agrégation

Avantage de mapReduce : reprise sur panne. Il y a un coordinateur qui dispatche les jobs entre les machines. Il est donc capable de reprendre un travail non terminé par une machine tombée et le donner à une autre.

## 16.7 Indexation

## 16.8 TP2

1. Importez le contenu du répertoire dans la base de données esipe2.

```
1 mongorestore --db=dbname path/to/file.bson
```

2. Combien avez-vous de collections dans la base ?

```
1 > show collections
```

3. Combien avez-vous d'enregistrements dans la collections ex ?

```
1 > db.ex.find().count()
```

4. Afficher le contenu du document dont l'id est 99.

```
1 > db.ex.find({_id:99}).pretty()
```

5. Afficher uniquement les items du document 99.

```
1 > db.ex.find({_id:99}, {items:1, _id:0}).pretty()
```

6. Afficher uniquement les quantités des items du document 99.

```
1 > db.ex.find({_id:99}, {"items.qte":1, _id:0}).pretty()
```

7. Combien y-a-t-il de documents dont le score est supérieur ou égale à 100

```
1 > db.ex.find({_id: {$gte:100}}).pretty().count()
```

### 16.8.1 Agrégation

Dans un premier temps, on utilise le framework d'agrégation de MongoDB. En utilisant la page [docs.mongodb.org](https://docs.mongodb.org), rédigez les requêtes suivantes :

1. La commande `distinct` permet d'obtenir d'avoir les données distincts d'une clé donnée. Écrire la requête permettant d'obtenir l'ensemble des scores de la collection `ex`.

```
1 > db.runCommand({"distinct" : "ex" , "key" : "score"})
```

2. Calculez la somme des scores pour l'ensemble des documents

```
1 > db.ex.aggregate([{$group: {_id:null, scoreTotal: {$sum: "$score"}}}])
2 {"_id" :null, "scoreTotal" :118650}
```

3. Calculez la somme des scores pour les documents dont les id sont inférieurs strictement à 500.

```
1 > db.ex.aggregate([{$match: {"_id":{$lt:500}}},{ $group: {_id:null, scoreTotal: {$sum: "$score"}}}])
2 {"_id" :null, "scoreTotal" :30030}
```

4. En agrégeant par score, calculez la somme des scores pour les documents et afficher uniquement les totaux supérieurs à 4000.

```
1 > db.ex.aggregate([{$group: {_id: "$score", scoreTotal: {$sum: "$score"}}},
2   {$sort: {scoreTotal: -1}}, {$match: {scoreTotal: {$gte:4000}}}]])
3 {"_id" :110,"scoreTotal" :18150}
4 {"_id" :100,"scoreTotal" :17400}
5 {"_id" :90,"scoreTotal" :16560}
6 {"_id" :70,"scoreTotal" :13790}
7 {"_id" :80,"scoreTotal" :13600}
8 {"_id" :60,"scoreTotal" :10920}
9 {"_id" :50,"scoreTotal" :9350}
10 {"_id" :40,"scoreTotal" :7880}
11 {"_id" :30,"scoreTotal" :5730}
```

5. Fournir la somme des quantités pour les éléments du tableau `items`.

```
1 > db.ex.aggregate([{$unwind: "$items" },{$group: {_id: "$items.prod", qte: {$sum: "$items.qte"}}}])
2 {"_id" : "B", "qte" :4028}
3 {"_id" : "A", "qte" :7050}
```

L'option `unwind` est nécessaire pour découpler les éléments d'un tableau afin de réaliser des opérations dessus.

### 16.8.2 MapReduce

La solution `map reduce` s'inspire de l'approche du framework du même nom de Google mais ne s'intègre pas directement avec ce dernier. Il y a des points en commun : elle peut être parallélisée sur plusieurs serveurs MongoDB, elle débute avec une étape de `map` sur chaque document de la collection, ensuite intervient une étape (`shuffle`) où les clés sont groupées pour émettre des listes de couples clé/valeur. Enfin l'étape de `reduce` utilise ces listes pour calculer un élément à partir des valeurs stockés. Cet élément est ensuite renvoyé au `shuffle` jusqu'à ce qu'un seul élément (le résultat) soit calculé. Dans MongoDB, MapReduce peut être utilisé pour calculer les opérations de `count`, `distinct`, `group` mais de manière moins efficace en performance que les opérations du framework d'agrégation. Les fonctions de `map` et `reduce` sont écrites en javascript.

N.B. : Il existe tout de même un connecteur Hadoop pour MongoDB.

On se base sur l'exemple suivant :

```
1 {"_id":1,favs:["spurs","knicks"]}
2 {"_id":2,favs:["lakers","spurs"]}
3 {"_id":3,favs:["knicks","celtics"]}
```

On veut compter le nombre d'occurrences dans le tableau `favs`.

1. Saisir le jeu d'essai dans la base `esipe2` et la collection `basket`

```
1 > db.basket.insertMany({...}, {...}, {...})
```

2. On va écrire une fonction javascript qui va émettre un couple clé/valeur pour chaque valeur rencontrée dans le tableau « favs » :

```

1 // fonction map
2 > map = function() {this.favs.forEach(function(f) {emit(f, {count:1});});}
3 // fonction emit
4 > function emit(k, v) {
5     print("emit");
6     print("k:" + k + "v:" + tojson(v));
7 }
8 // test
9 > map.apply(db.basket.findOne())

```

3. Maintenant que l'on a vu le principe de la fonction map, on va écrire la fonction reduce. Celle-ci prend en entrée des couples clé/valeurs et retourne une clé/valeur. Dans notre exemple, la fonction reduce sera la suivante :

```

1 > reduce = function(key, values) {
2     total = 0;
3     for (var i=0;i<values.length;i++) {
4         total += values[i].count;
5     }
6     return {"count" :total};
7 }
8 // test
9 > reduce("celtics", [{count:1},{count:2}])

```

4. On va maintenant tester au sein du framework mapreduce en utilisant nos fonctions `map` et `reduce` :

```

1 res= db.basket.mapReduce(map,reduce,{out:{inline:1}})

```

l'option `out` peut prendre plusieurs valeurs. Ici `inline:1` indique qu'aucune collection ne sera créée à la suite de l'opération et donc que le résultat sera stocké en RAM (le résultat doit faire moins de 16Mo). Autres options : `{replace : "nomCollection"}` pour stocker le résultat dans une collection, `merge` (si recouvrement de clés, la nouvelle valeur écrase l'ancienne et `reduce` (relance un `reduce` si dans l'ancien collection, il existait un clé identique). On pouvait aussi utiliser la syntaxe :

```

1 > db.runCommand({mapreduce : "basket", "map":map, "reduce":reduce})

```

5. Fournir en utilisant l'approche mapReduce le nombre d'occurrences par score dans la collection `ex` de la base de données `esipe2`.

```

1 > mapScore = function() {emit(this.score, {count:1});}
2 // on réutilise la fonction reduce
3 > res = db.ex.mapReduce(mapScore,reduce,{out:{inline:1}})

```

6. Vérifier le résultat avec une requête exploitant le framework d'agrégation

```

1 > db.ex.aggregate([{$group: {_id: "$score", scoreTotal: {$sum: 1}}},{$sort: {_id:1}}])
2 {"_id" :10,"scoreTotal" :179}
3 {"_id" :20,"scoreTotal" :174}
4 {"_id" :30,"scoreTotal" :191}
5 {"_id" :40,"scoreTotal" :197}
6 {"_id" :50,"scoreTotal" :187}
7 {"_id" :60,"scoreTotal" :182}
8 {"_id" :70,"scoreTotal" :197}
9 {"_id" :80,"scoreTotal" :170}
10 {"_id" :90,"scoreTotal" :184}
11 {"_id" :100,"scoreTotal" :174}
12 {"_id" :110,"scoreTotal" :165}

```

### 16.8.3 Indexation

Il est possible de créer des indexes à l'aide de l'instruction `createIndex(champ:1)` sur une collection donnée. La valeur 1 peut être remplacée par -1, indiquant le sens de l'indexation (asc vs desc). L'instruction `explain()` va nous permettre d'observer l'impact de l'utilisation d'un index lors de l'exécution d'une requête :

```

> db.collection.find(...).explain()

```

1. Écrire une requête permettant d'obtenir les documents dont le score est 30. Lancer la même requête avec un `explain()`.
2. Ajouter un index à collection `ex` sur la clé `score` et lancer la requête précédente avec un `explain()`. Comparez les résultats.

```
1 > db.ex.createIndex({score:1})
```

3. Il est possible de créer un index sur les clés d'un tableau. Créer un index sur la quantité (`qte`) du tableau `item`. Imaginez une requête permettant de mettre en évidence le gain procuré par cet index.

```
1 > db.ex.createIndex({"items.qte":1})
```

## 16.9 Administration – API Java

### 16.9.1 Administrer une base de données MongoDB

**A.** A titre d'exemple, on va créer une nouvelle base de données sur le domaine de la formation (on nomme la base de données `forma`). Elle comportera des informations sur des enseignants (avec nom, prénom et numéro de mobile) et des informations sur les cours (nom, durée (en jours)). On veut pouvoir obtenir les informations sur l'enseignant donnant un cours. On part du principe que chaque enseignant peut donner beaucoup de cours et qu'il y a de nombreuses opérations d'écriture (rapport écriture/lecture sur la base est proche de 1), c'est-à-dire que les enseignants d'un cours peuvent être modifiés régulièrement. Sur ce principe, on considère une modélisation avec 2 collections : `cours` et `enseignant`.

1. Proposer une solution pour renseigner l'enseignant d'un cours.
2. Renseigner 2 documents pour les enseignants et 2 documents pour les cours. Faire en sorte que chaque enseignant enseigne cours.

```
1 > db.enseignant.insertMany(
2   { "lastname" : "Cure", "firstname" : "Olivier", "tel" : "0623625288" },
3   { "lastname" : "Broussole", "firstname" : "Adrien", "tel" : "0623625288" }
4 )
5 > db.cours.insertMany(
6   { "name" : "Scala", "duree" : 4, "enseignantId" : ObjectId("592fc60fd2b3865faa63c807") },
7   { "name" : "MongoDB", "duree" : 3, "enseignantId" : ObjectId("592fc5ccd2b3865faa63c806") }
8 )
```

3. Écrire le document permettant d'obtenir le nom et le prénom de l'enseignant depuis un cours donné (sans saisir vous-même de constantes, sauf celle associée au nom du cours).

```
1 > db.cours.aggregate([{$match: {name: "Scala"}},
2   {$lookup: {from: "enseignant", localField: "enseignantId", foreignField: "_id", as: "enseignantInfo"}}])
```

L'option `$lookup` permet d'utiliser un élément d'une collection et de faire un `find` sur une autre collection.

N.B. : l'attribut `as` crée une nouvelle collection pour stocker les informations. Il faut donner un nom différent entre la valeur `from` et `as` sinon la deuxième collection remplacera la première.

4. Faire une sauvegarde de la base de données (avec ces 2 collections et ces documents)

```
1 mongodump --db=forma
```

5. Regarder les métadonnées des dumps

6. Supprimer un enseignant puis supprimer les 2 cours. Que remarquez-vous par rapport à une base de données relationnelle.

```
1 > db.enseignant.remove({lastname: "Broussole"})
2 > db.cours.remove({name: "Scala"})
3 > db.cours.remove({name: "MongoDB"})
```

7. Faites une vérification des collections de la base de données : existence et nombre de documents.

```
1 > db.enseignant.find()
2 > db.cours.find()
```

8. Faire une restauration complète et propre (option `--drop`) et faire une vérification que vous avez bien 2 documents par collection. Voyez-vous un problème ?

```
1 > mongorestore --db=forma --drop path/to/file.bson
```

Le problème est qu'il ne garde pas la liaison entre les cours et les enseignants. En effet, les id des enseignants sont écrits à la main pour chaque cours et lors de l'insert d'un enseignant, l'id est généré par mongo. La restauration de la base engendre la génération d'un nouvel id.

#### B. On va importer un jeu de données CSV dans une nouvelle base de données.

1. Importer, dans la base de données `moviedb`, les informations sur les films du fichier `movies.csv`. La première ligne correspond aux champs du fichier. Les informations seront stockées dans la collection `movie`.

```
1 mongoimport --db=moviedb --collection=movie --type=csv --headerline movies.csv
```

2. Créer un utilisateur admin avec les droits en lecture/écriture sur la base admin et la capacité à administrer l'ensemble des bd.

```
1 > use admin
2 switched to db admin
3 > db.createUser({user:'admin',pwd:'pass', roles:[{role:'readWriteAnyDatabase',db:'admin'},
4 {role:'userAdminAnyDatabase',db:'admin'}]})
```

Relancer le serveur avec l'option authentication, puis créer un utilisateur avec les droits en lecture seulement sur la base de données movies.

```
1 > db.createUser({user:'userMovie',pwd:'readOnly', roles:[{role:'read',db:'moviedb'}]})
2 > use admin
3 > db.auth('', '')
4 > db.movie.insert({title:"Star Wars VII"})
5 WriteResult({
6   "writeError" : {
7     "code" : 13,
8     "errmsg" : "not authorized on moviedb to execute command { insert: \"movie\", documents: ... , ordered: true }"
9   }
10 })
```

Relancez la console et connectez vous avec ce nouveau utilisateur. Faire une lecture puis une écriture.

```
1 mongod --dbpath /home/inti/13-mongodb/data-mongodb/ --auth
2 mongo
3 > use admin
4 switched to db admin
5 > db.auth("admin","pass")
6 1
```

3. Que se passe-t-il si on relance `mongod` sans l'option `--auth` ?  
On accède à toutes les db en mode lecture et écriture.

#### C. On va travailler à nouveau sur la base `moviedb` mais en exploitant les données depuis un programme Java. On fait créer un projet Maven et insérer la dépendance suivante dans le

pom.xml

```
1 <dependency>
2   <groupId>org.mongodb</groupId>
3   <artifactId>mongo-java-driver</artifactId>
4   <version>3.4.2</version>
5 </dependency>
```

En vous aidant de la javadoc (lien web), vous allez créer une première classe Java qui doit :

1. créer une connexion MongoClient sur localhost et le port 27017
2. se connecter à la base de données `moviedb`
3. se connecter à la classe `movies` et afficher, sous la forme d'un document JSON, le premier enregistrement de cette collection.

4. Obtenir le nombre de documents dans cette collection `moviedb`
5. Fournir la liste des films dont l'année de parution est 1995 en utilisant un cursor (`MongoCursor`).
6. Insérer un nouveau film (« Logan » sorti en 2017) en enrichissant votre programme Java
7. Mettre à jour le film Logan en indiquant que sa `release_date` est le 03-03-2017.
8. Supprimer le film Logan

```

1  import com.mongodb.*;
2  import com.mongodb.client.FindIterable;
3  import com.mongodb.client.MongoCollection;
4  import com.mongodb.client.MongoCursor;
5  import com.mongodb.client.MongoDatabase;
6  import org.bson.Document;
7  /**
8   * Created by inti on 31/05/17.
9   */
10 public class Main {
11     public static void main(String args[] ) {
12         MongoClient mongoClient = new MongoClient("localhost", 27017);
13         MongoDatabase db = mongoClient.getDatabase("moviedb");
14
15         MongoCollection<Document> coll = db.getCollection("movie");
16         FindIterable<Document> iterable = coll.find();
17         iterable.noCursorTimeout(true);
18         Document filtre = new Document();
19         filtre.put("release_year",1995);
20         iterable.filter(filtre);
21         // cursor.limit(1);
22         MongoCursor c = iterable.iterator();
23         int i = 1;
24         while (c.hasNext()) {
25             // System.out.println("Inserted Document: "+i);
26             Document res = (Document) c.next();
27             System.out.println("titre: "+res.get("title").toString());
28             // System.out.println(res);
29             i++;
30         }
31         System.out.println(i);
32         System.out.println(coll.count());
33
34         Document newMovie = new Document();
35         newMovie.put("title","Logan");
36         newMovie.put("release_year",2017);
37
38         coll.insertOne(newMovie);
39
40         Document search = new Document();
41         search.put("title","Logan");
42         FindIterable<Document> result = coll.find(search);
43         System.out.print(result.first());
44         search = result.first();
45         search.put("release_date","03-03-2017");
46         coll.findOneAndReplace(newMovie,search);
47         // coll.findOneAndDelete(search);
48     }
49 }

```

## 16.10 Réplication

```

1  SECONDARY> db.entries.find()
2  { "_id" : ObjectId("593018d431973b1735980bbf"), "name" : "Selena", "age" : 1 }
3  { "_id" : ObjectId("593018dd31973b1735980bc0"), "name" : "Angela", "age" : 27 }
4  SECONDARY> db.entries.insert({name:"Samuel",age:27})
5  WriteResult({ "writeError" : { "code" : 10107, "errmsg" : "not master" } })

```

## 16.11 CAP

- Consistency : deux requêtes simultanée sur deux machines différentes doivent retourner le même résultat.

- Availability : tout doit être disponible.
- Partitioning : gestion de la panne réseaux entre plusieurs machines.

## 16.12 Verrou

MongoDB crée un verrou en écriture et laisse libre la lecture. Il ne peut pas y avoir d'accès en écriture simultanée sur une ressource. C'est pour cela que Mongo n'est pas Availability.

## 16.13 Évaluation

Vous devez charger les données dans une base de données qu'on nommera drugs. Pour les scripts compo, cis et cis\_cip, vous générerez respectivement les collections compo, cis et ciscip. À la suite de ce chargement, vous devez fournir :

1. le nombre de documents dans la collection compo. (symbole " modifié en ' dans les fichiers txt)

```

1 mongoimport --db=drugs --drop --collection=cis --type=tsv
2   -f cis,denom,forme,voie,statadm,typproc,etacom,datam,labotit,numEU,titulaire,surveillance
3   drugs/cis.txt
4 # 2017-06-01T13:51:16.992+0200 connected to: localhost
5 # 2017-06-01T13:51:16.993+0200 dropping: drugs.cis
6 # 2017-06-01T13:51:17.553+0200 imported 14527 documents
7 mongoimport --db=drugs --drop --collection=ciscip --type=tsv
8   -f cis,cip7,pres,statadm,etacom,datedecl,cip13,agrcol,taux,prix,prixtot,addprix,indications
9   drugs/cis_cip.txt
10 # 2017-06-01T14:07:28.384+0200 connected to: localhost
11 # 2017-06-01T14:07:28.384+0200 dropping: drugs.ciscip
12 # 2017-06-01T14:07:28.996+0200 imported 20171 documents
13 mongoimport --db=drugs --drop --collection=compo --type=tsv
14   -f cis,element,codesub,nomsub,dosage,ref,nature,fraction
15   drugs/compo.txt
16 # 2017-06-01T14:09:07.344+0200 connected to: localhost
17 # 2017-06-01T14:09:07.344+0200 dropping: drugs.collection=compo
18 # 2017-06-01T14:09:07.902+0200 imported 27176 documents

```

2. Fournir le nombre d'entrées dans la collection compo dont le code de substance est 2202.

```

1 > db.compo.count({codesub:2202})
2 247

```

3. Fournir le nom de substance correspond au code substance 2202 (ne sortir qu'une seule ligne).

```

1 > db.compo.findOne({codesub: 2202},{nomsub:1, _id:0})
2 {"nomsub" : "PARACETAMOL" }

```

4. À l'aide d'une opération 'aggregate', fournir la liste des désignations de la collection compo.

```

1 > db.compo.aggregate({$group: {_id: "$element"}})

```

5. Enrichir la requête 4 pour avoir en plus le nombre d'occurrences de chaque désignations

```

1 > db.compo.aggregate({$group: {_id: "$element", total: {$sum: 1}}})

```

6. Modifier le requête 5 pour n'obtenir que la désignation 'gel'.

```

1 > db.compo.aggregate([
2   {$group: {_id: "$element", total: {$sum: 1}}},
3   {$match: {"_id": "gel"}}
4 ])
5 {"_id" : "gel", "total" :295}

```

7. Compléter la requête 6 pour obtenir les désignations 'gel' ou 'pommade'

```

1 > db.compo.aggregate([
2   {$group: {_id: "$element", total: {$sum: 1}}},
3   {$match: {$or :[{"_id": "gel"}, {"_id": "pommade"}]}}

```

```

4         ])
5     {"_id": "pommade", "total": 707}
6     {"_id": "gel", "total": 295}

```

8. Compléter la requête 7 pour n'obtenir que les produits dont le code substance est 2202.

```

1 > db.compo.aggregate([
2     {$match: {"codesub": 2202}},
3     {$group: {_id: "$element", total: {$sum: 1}}},
4     {$match: {$or: [{"_id": "gel"}, {"_id": "pommade"}]}}
5 ])
6 Aucun resultat fourni puisque le code sub 2202 ne correspond qu'a du gel ou des pommade

```

9. Modifier la requête 8 en remplaçant 'gel' et 'pommade' par 'solution' et 'poudre'

```

1 > db.compo.aggregate([
2     {$match: {"codesub": 2202}},
3     {$group: {_id: "$element", total: {$sum: 1}}},
4     {$match: {$or: [{"_id": "solution"}, {"_id": "poudre"}]}}
5 ])
6 {"_id": "poudre", "total": 29}
7 {"_id": "solution", "total": 13}

```

10. Calculer la moyenne du prix total des produits de la collection ciscip

```

1 > db.ciscip.aggregate([{$group: {_id: null, avgPrixTot: {$avg: "$prixTot"}}}])

```

11. Afficher uniquement le cip7 et le prixTot depuis la collection ciscip dont les prix sont au minimum 1.00 euros et maximum 1.98 euros exclus.

```

1 > db.ciscip.find(
2     {$and: [{prixTot: {$gt: 1.0}}, {prixTot: {$lt: 1.98}}]},
3     {cip7: 1, prixTot: 1, _id: 0})

```

12. Afficher l'ensemble des indexes de la collection ciscip.

```

1 > db.ciscip.getIndexes()

```

13. Modifier la requête 12 pour afficher également l'état de commercialisation (comm) et le taux de remboursement de la sécurité sociale (tauxss).

```

1 > db.ciscip.find(
2     {$and: [{prixTot: {$gt: 1.0}}, {prixTot: {$lt: 1.98}}]},
3     {cip7: 1, prixTot: 1, _id: 0, taux: 1, etacom: 1})

```

14. Modifier la requête 14 pour ne faire apparaître que les produits dont le taux de remboursement est de 15%

```

1 > db.ciscip.find(
2     {$and: [
3         {prixTot: {$gt: 1.0}},
4         {prixTot: {$lt: 1.98}},
5         {taux: {$eq: "15%"}}
6     ]},
7     {cip7: 1, prixTot: 1, _id: 0, taux: 1, etacom: 1})

```

15. Afficher le plan d'exécution de la requête 14

```

1 > db.ciscip.find(
2     {$and: [{prixTot: {$gt: 1.0}}, {prixTot: {$lt: 1.98}}, {taux: {$eq: "15%"}}]},
3     {cip7: 1, prixTot: 1, _id: 0, taux: 1, etacom: 1})
4 .explain()

```

16. L'exécution de la requête 14 exploite-t-elle un index ? Justifier.

> Le explain() nous indique que la requete est faite en "stage" : "COLLSCAN". On scan sur toutes les valeurs des colonnes cibles. Il n'y pas d'utilisation d'index. 17. Modifier l'affichage de la requête 14 de manière à avoir les résultats triés dans l'ordre décroissant des valeurs de prixTot

```

1 > db.ciscip.find(
2     {$and: [{prixTot: {$gt: 1.0}}, {prixTot: {$lt: 1.98}}, {taux: {$eq: "15%"}}]},
3     {cip7: 1, prixTot: 1, _id: 0, taux: 1, etacom: 1})
4 .sort({prixTot: -1})

```



18. Supprimer le champ taxe du médicament dont le cis est 60085578

```
1 > db.ciscip.update({cis:60085578}, {$unset: {"tax": ""}})
```

# Chapitre 17

## TP Spark

build.sbt

```
1 name := "Spark Project"
2
3 version := "1.0"
4
5 scalaVersion := "2.11.8"
6
7 libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.0"
8 libraryDependencies += "org.scalatest" %% "scalatest" % "2.2.6"
9 libraryDependencies += "com.typesafe.play" % "play-json_2.11" % "2.4.3"
10
11 /* resolvers += "Typesafe Repo" at "http://repo.typesafe.com/typesafe/release/" */
12 /* resolvers += "sbt-idea-repo" at "http://mpeltonen.github.com/maven/" */
```

Afin de pouvoir utiliser play-json avec spark, il faut des versions compatibles. On peut prendre play-json\_2.11 % 2.4.3 ou 2.4.10 ou 2.4.0.

Récupérer un JSON sous spark-shell

```
1 > val pathToFile = "11-spark/spark-lab/data/reduced-tweets.json"
2 > val df = spark.read.json(pathToFile)
3 > case class Tweet (
4     id :String,
5     user :String,
6     text :String,
7     place :String,
8     country :String)
9 > val tweets = sc.makeRDD(df.collect())
10     .map(x => Tweet(x(1).toString,x(4).toString,x(3).toString,x(2).toString,x(0).toString))
```

Script scala

```
1 import com.tp.spark.utils.TweetUtils.Tweet
2 import org.apache.spark.{SparkConf, SparkContext}
3 import play.api.libs.json.Json
4 import scala.collection.Map
5
6 object Ex4InvertedIndex {
7   /**
8    *
9    * Buildind a hashtag search engine
10    * The goal is to build an inverted index. An inverted is the data structure used to build search engines.
11    * How does it work?
12    * Assuming #spark is an hashtag that appears in tweet1, tweet3, tweet39.
13    * The inverted index that you must return should be a Map (or HashMap) that contains a
14    * (key, value) pair as (#spark, List(tweet1,tweet3, tweet39)).
15    *
16    */
17   case class Tweet (
18       id :String,
19       user :String,
20       text :String,
21       place :String,
```

```

22         country :String
23     )
24     def invertedIndex(): Map[String, Iterable[Tweet]] = {
25         // create spark configuration and spark context
26         val conf = new SparkConf ()
27             .setAppName ("Inverted index")
28             .setMaster ("local[*]")
29         val sc = SparkContext.getOrCreate(conf)
30         sc.textFile ("data/reduced-tweets.json")
31             .map(loadJson(_))
32             .flatMap(hashtag(_))
33             .reduceByKey(_+_ )
34             .collectAsMap()
35     }
36     def loadJson(texte: String): Tweet = {
37         implicit val tweetFormat = Json.format[Tweet]
38         Json.parse(texte).as[Tweet]
39     }
40     def hashtag(testTweet: Tweet) = {
41         testTweet.text
42             .split(" ")
43             .filter(x => (x.startsWith("#") && x.size>1))
44             .map(x => (x, List(testTweet)))
45     }
46 }

```

La fonction `loadJson` est doit être défini à l'extérieur pour ensuite l'utiliser dans un `map`. Dans le cas contraire, le compilateur plante avec l'erreur `Task not serializable`. Spark est conçu pour paralléliser les opérations.

### 17.0.1 Self-Contained Applications

Suppose we wish to write a self-contained application using the Spark API. We will walk through a simple application in Scala (with sbt), Java (with Maven), and Python. We'll create a very simple Spark application in Scala-so simple, in fact, that it's named `SimpleApp.scala` :

SimpleApp.scala

```

1  import org.apache.spark.SparkContext
2  import org.apache.spark.SparkContext._
3  import org.apache.spark.SparkConf
4
5  object SimpleApp {
6      def main(args: Array[String]) {
7          val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
8          val conf = new SparkConf().setAppName("Simple Application")
9          val sc = new SparkContext(conf)
10         val logData = sc.textFile(logFile, 2).cache()
11         val numAs = logData.filter(line => line.contains("a")).count()
12         val numBs = logData.filter(line => line.contains("b")).count()
13         println(s"Lines with a: $numAs, Lines with b: $numBs")
14         sc.stop()
15     }
16 }

```

Note that applications should define a `main()` method instead of extending `scala.App`. Subclasses of `scala.App` may not work correctly.

This program just counts the number of lines containing 'a' and the number containing 'b' in the Spark README. Note that you'll need to replace `YOUR_SPARK_HOME` with the location where Spark is installed. Unlike the earlier examples with the Spark shell, which initializes its own `SparkContext`, we initialize a `SparkContext` as part of the program.

We pass the `SparkContext` constructor a `SparkConf` object which contains information about our application.

Our application depends on the Spark API, so we'll also include an sbt configuration file, `build.sbt`, which explains that Spark is a dependency. This file also adds a repository that Spark depends on :

build.sbt

```

1  name := "Simple Project"

```

```
2
3 version := "1.0"
4
5 scalaVersion := "2.11.7"
6
7 libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.1"
```

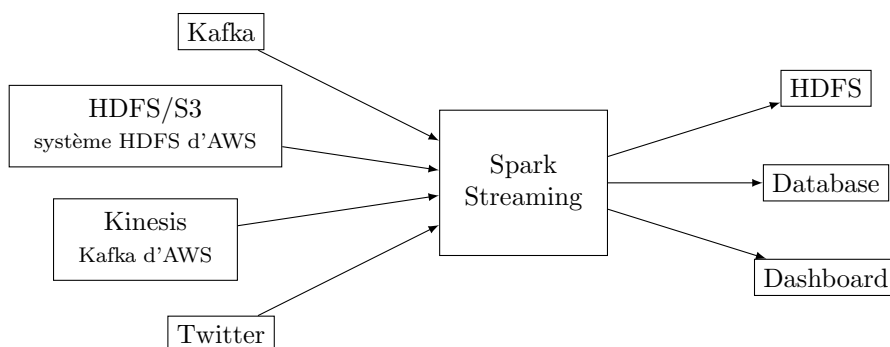
For sbt to work correctly, we'll need to layout SimpleApp.scala and build.sbt according to the typical directory structure. Once that is in place, we can create a JAR package containing the application's code, then use the spark-submit script to run our program.

```
1 # Your directory layout should look like this
2 $ find .
3 .
4 ./build.sbt
5 ./src
6 ./src/main
7 ./src/main/scala
8 ./src/main/scala/SimpleApp.scala
9
10 # Package a jar containing your application
11 $ sbt package
12 ...
13 [info] Packaging {..}/{..}/target/scala-2.11/simple-project_2.11-1.0.jar
14
15 # Use spark-submit to run your application
16 $ YOUR_SPARK_HOME/bin/spark-submit \
17   --class "SimpleApp" \
18   --master local[4] \
19   target/scala-2.11/simple-project_2.11-1.0.jar
20 ...
21 Lines with a: 46, Lines with b: 23
```

# Chapitre 18

## Spark Streaming

@Adrien Broussolle



### 18.1 Introduction

Aujourd'hui, on n'arrive pas à avoir la gestion de données BigData et de temps réels. traitement temps quasi-réel = quelques secondes :

- social network :
- statistic/metrics : on veut être en courant dans la seconde s'il y a un pic de connexion sur un site web pour éventuellement allumer un autre serveur pour gérer la charge.
- fraud/intrusion detection : avertissement en temps réel d'une fraude bancaire afin de faire opposition au plus vite.
- paiement online
- login

#### 18.1.1 Event Sourcing

Au lieu de maintenir des états, on écrit les changements.

- on écoute les événements.
- append-only : on n'a plus le problème d'écriture concurrente.
- bug and crash resilient : on a la possibilité d'isoler un événement qui a causé un bug et ainsi retrouver un état cohérent en rejouant tous les événements corrects.
- space consuming : étant donnée que l'on autorise pas la modification. Chaque événement même petit génère de la données.

Solution : on associe le streaming avec une gestion d'états i.e. au bout d'un certain temps de streaming, un batch agrège toutes les données pour les sauvegarder.

**Architecture Lambda :** est un processus qui associe les avantages des systèmes batch et streaming. Il est capable de gérer en streaming une quantité massive de données à la seconde et de gérer du batch lors d'une transaction unique mais volumineuse.

Pré-requis :

- scalable
- scalable à la seconde : dès qu'un serveur a plus de 50% de charge, on démarre une autre instance. Dès qu'on est inférieur à 30%, on en éteins une.
- simple : chaque ingénieur doit savoir gérer le batch et le streaming

## 18.2 Spark Streaming

C'est une extension de l'API Spark qui fournit un processus de Streaming. La données peut après être engrenagée dans Kafka, Flume, ZeroMQ, Kinesis, TCP sockets...

**Réseaux :**

- TCP : acquittement i.e. avec accusé de réception
- UDP : pas d'acquittement

### 18.2.1 Principe

- streaming en appliquant des micro-batch
- 500ms à 1s de latence
- chaque batch est composé de RDD
- batch et micro-batch peuvent être combinés dans une même instance

### 18.2.2 Transformation

- RDD :
  - map, flatMap, filter, ...
  - countByValue (fréquence d'apparition) au lieu de countByKey
- transformation et sauvegarde d'états :
  - updateStateByKey
  - reduceByKeyAndWindow, countByValueAndWindow, ...

### 18.2.3 Action

- print
- saveAs
  - saveAsTextFiles
  - saveAsObjectFiles
  - saveAsHadoopFiles
- foreachRDD

**Streaming Context**

- un seul par JVM
- crée par SparkConf ou SparkContext et définie par l'intervalle de batch
- évaluation lazy
- ssc.start() : pour déclencher le traitement
- ssc.awaitTermination() : pour signaler la fin des processus

## Exemple

```

1 val sc = new SparkConf()
2 val ssc = new StreamingContext(sparkConf, Seconds(1))
3 // args(0)=IP et args(1)=port
4 val lines = ssc.socketTextStream(args(0), args(1).toInt, StorageLevel.MEMORY_AND_DISK_SER)

```

## 18.3 TP

Nous allons associer Spark Streaming avec Kafka cf. <http://spark.streaming-kafka>.

## build.sbt

```

1 name := "pop-test"
2
3 version := "0.1"
4
5 scalaVersion := "2.11.8"
6
7 libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.0" % "provided"
8 libraryDependencies += "joda-time" % "joda-time" % "2.9.9"
9 libraryDependencies += "org.apache.spark" % "spark-streaming_2.11" % "2.1.1"
10 libraryDependencies += "org.apache.spark" % "spark-streaming-kafka-0-10_2.11" % "2.1.1"
11 libraryDependencies += "com.typesafe.play" % "play-json_2.11" % "2.4.3"
12
13 /* resolvers += Seq(
14   "Typesafe Repository" at "http://repo.typesafe.com/typesafe/releases/",
15   "Confluent" at "http://packages.confluent.io/maven/"
16 ) */

```

### 18.3.1 Objet compagnon

Lors d'un parsing, il est judicieux de définir le format de désérialisation pour chaque `case class`. Pour ce faire, il est recommandé de le définir dans un `object` portant le même nom que la `case class`. De cette manière, ce qui est défini dans l'objet est accessible pour toutes les instances du programme et pourra être chargé lors de l'utilisation de la `case class`. Par convention, on crée un fichier par `case class`.

## case class + object compagnon

```

1 import org.joda.time.DateTime
2 import play.api.libs.json.Json
3 //auto generated code by avro4s
4 case class PlayLogHeader(
5   version: Int,
6   makerId: String,
7   envId: Option[Int],
8   id: String,
9   creationTime: DateTime
10 )
11 object PlayLogHeader {
12   implicit val playLogHeaderFormat = Json.format[PlayLogHeader]
13 }

```

Dans certain cas, la bibliothèque `JodaTime` ne reconnaît pas nativement le format de date utilisé. Il est alors nécessaire de redéfinir la façon de lire et d'écrire le format voulu.

## Ajout d'un format de date à JodaTime

```

1 object PlayLogBody {
2   val pattern = "yyyy-MM-dd'T'HH:mm:ssZ"
3   val otherPattern = "yyyy-MM-dd'T'HH:mm:ss"
4   val anotherPattern = "yyyy-MM-dd"
5   implicit val dateReadFormat = Reads.jodaDateReads(pattern)
6     .orElse(Reads.jodaDateReads(otherPattern))
7     .orElse(Reads.jodaDateReads(anotherPattern))
8   implicit val dateWriteFormat = Writes.jodaDateWrites(pattern)
9   implicit val playLogHeaderFormat = Json.format[PlayLogHeader]
10 }

```

## Script Test

```

1 package model.domain
2
3 import org.joda.time.DateTime
4 import org.scalatest.{FunSuite, Matchers}
5 import play.api.libs.json.Json
6
7 class JsonSpec extends FunSuite with Matchers {
8   test("should count the number of couple (user, tweets)") {
9     val jsonString =
10       """{
11         "header":{
12           "version":2,
13           "makerId":"LeChat-PPR",
14           "id":"d7d35fa9-0461-4d6d-bc76-e7a879e1f887",
15           "creationTime":"2017-03-29T13:54:04+0200"
16         }
17       ,
18       "body":{
19         "playerid":"35776754",
20         "hostname":"fr-lab-lx0014.jcd.priv",
21         "playertype":"dms-player",
22         "timestamp":"2017-03-29T15:46:59+0200",
23         "duration":10,"creativeid":"102675169",
24         "frameid":"50732058",
25         "media":[],
26         "condition":[],
27         "context":[]
28       }
29     }
30     """
31     val
32     expectedBody = PlayLogBody(
33       playerid = "35776754",
34       hostname = Some("fr-lab-lx0014.jcd.priv"), playertype = "dms-player",
35       timestamp = new DateTime("2017-03-29T15:46:59+0200"),
36       campaignid = None,
37       campaignname = None,
38       duration = Some(10),
39       creativeid = "102675169",
40       frameid = Some("50732058"),
41       status = None,
42       media = Nil,
43       condition = Nil,
44       context = Nil
45     )
46     val expectedHeader = PlayLogHeader(
47       envId = None,
48       version = 2,
49       makerId = "LeChat-PPR",
50       id = "d7d35fa9-0461-4d6d-bc76-e7a879e1f887",
51       creationTime = new DateTime("2017-03-29T13:54:04+0200")
52     )
53     val js = Json.parse(jsonString)
54     println(PlayLog.playLogFormat.reads(js))
55     PlayLog.playLogFormat.reads(js).isSuccess should be(true)
56     PlayLog.playLogFormat.reads(js).get.header should be(expectedHeader)
57     PlayLog.playLogFormat.reads(js).get should be(
58       PlayLog(
59         expectedHeader,
60         expectedBody
61       )
62     )
63   }
64 }

```

## Imports

```

1 import model.domain.{Campaign, PlayLog, PlayLogBody}
2 import org.apache.spark.{SparkConf, SparkContext}
3 import org.apache.spark.streaming.{Seconds, StreamingContext}
4 import org.apache.kafka.clients.consumer.ConsumerRecord
5 import org.apache.kafka.common.serialization.StringDeserializer
6 import org.apache.spark.broadcast.Broadcast

```



```

7 import org.apache.spark.streaming.dstream.{DStream, InputDStream}
8 import org.apache.spark.streaming.kafka010._
9 import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
10 import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe
11 import org.joda.time.DateTime
12 import play.api.libs.json.{JsError, JsSuccess, Json}

```

## Main

```

1 object Main extends App {
2
3   import MainSupport._
4
5   val conf = new SparkConf().setMaster("local[*]").setAppName("SparkStreaming")
6   val ssc = new StreamingContext(conf, Seconds(5))
7
8   val kafkaParams = Map[String, Object](
9     "bootstrap.servers" -> "localhost:9092",
10    "key.deserializer" -> classOf[StringDeserializer],
11    "value.deserializer" -> classOf[StringDeserializer],
12    "group.id" -> "use_a_separate_group_id_for_each_stream",
13    "auto.offset.reset" -> "latest",
14    "enable.auto.commit" -> (false: java.lang.Boolean)
15  )
16  val topics = Array("jsonstream")
17  val kafkaStream: InputDStream[ConsumerRecord[String, String]] = KafkaUtils.createDirectStream[String, String](
18    ssc,
19    PreferConsistent,
20    Subscribe[String, String](topics, kafkaParams)
21  )
22  val dateLimit: DateTime = new DateTime("2017-03-28T12:00:00+0200")
23
24  val campaigns: Broadcast[Map[String, Campaign]] = loadCampaign(conf)
25
26  val stream: DStream[PlayLogBody] = kafkaStream.map(_.value())
27    .flatMap(StringToPlayLog(_))
28    .filter(_.header.creationTime.isAfter(dateLimit))
29    .map(checkPlayLog(_, campaigns))
30    .coalesce(1)
31    .saveAsTextFiles("enriched-pop.json")
32
33  stream.map(getInterest(_))
34    .reduceByKey((x, y) => sumOption(x, y))
35    .coalesce(1)
36    .saveAsTextFiles("aggregate-pop.json")
37
38  ssc.start()
39  ssc.awaitTermination()
40  //}
41 }

```

## Support

```

1 object MainSupport {
2   val campaignFile = "campaign.csv"
3   def getPlayLog(consumerValue: String): PlayLog = Json.parse(consumerValue).as[PlayLog]
4   def StringToPlayLog(consumerValue: String): Option[PlayLog] = {
5     Json.parse(consumerValue).validate[PlayLog] match {
6       case JsError(e) => println(e); None
7       case JsSuccess(t, _) => Some(t)
8     }
9   }
10  def checkPlayLog(playLog: PlayLog, campaigns: Broadcast[Map[String, Campaign]]): PlayLogBody = {
11    val campaign = campaigns.value.get("47820904").get
12    if (playLog.body.campaignid.isEmpty) {
13      PlayLogBody(
14        playLog.body.playerid,
15        playLog.body.hostname,
16        playLog.body.playertype,
17        playLog.body.timestamp,
18        Some(campaign.campaignId),
19        Some(campaign.campaignName),
20        playLog.body.duration,
21        playLog.body.creativeid,

```

```

22     playLog.body.frameid,
23     playLog.body.status,
24     playLog.body.media,
25     playLog.body.condition,
26     playLog.body.context
27 )
28 }
29 else {
30     playLog.body
31 }
32 }
33 def loadCampaign(conf: SparkConf): Broadcast[Map[String, Campaign]] = {
34     val sc = SparkContext.getOrCreate(conf)
35     val campaigns: Map[String, Campaign] = sc.textFile(getClass.getClassLoader.getResource(campaignFile).getPath)
36       .flatMap(_.split("\n"))
37       .map(line => parseCampaign(line))
38       .map(x => (x.campaignId, x))
39       .collect()
40       .toMap
41     sc.broadcast(campaigns)
42 }
43 def parseCampaign(campaign: String): Campaign = {
44     val elems = campaign.split(",")
45     Campaign(elems(0).toString, elems(1).toString)
46 }
47 def getInterest(playLogBody: PlayLogBody): ((String, Option[String], Option[String], String), Option[Int]) = {
48     ((playLogBody.creativeid, playLogBody.frameid, playLogBody.campaignid, playLogBody.playerid), playLogBody.duration)
49 }
50 def sumOption[T: Numeric](x: Option[T], y: Option[T]): Option[T] = {
51     val numeric = implicitly[Numeric[T]]
52     val list = List(x, y)
53     list.foldLeft(Option(numeric.zero)) {case (acc, el) =>
54         if (el.isEmpty) {
55             acc
56         }
57         else {
58             el.flatMap(value => acc.map(ac => numeric.plus(ac, value)))
59         }
60     }
61 }
62 }

```

Pour envoyer des données à Kafka depuis un fichier cf. [10.12](#)

# Chapitre 19

## Spark SQL

- utilisation de requête SQL
- lecture depuis une table Hive
- exécution sous forme de DataSet et/ou DataFrame

### 19.1 DataFrames

- un DataFrame

Doc

Le paquet `spark-shell` initie plusieurs variable au lancement :

```
1 Spark context Web UI available at http://192.168.1.143:4040
2 Spark context available as 'sc' (master = local[*], app id = local-1496908390469).
3 Spark session available as 'spark'.
```

#### 19.1.1 JSON

Chargement de JSON

```
1 > val etudiants1Df = spark.read.json("etudiants1.json")
2 > val etudiants2Df = spark.read.json("etudiants2.json")
3 > val etuDf = etudiants1Df.union(etudiants2Df)
4 > etuDf.printSchema()
```

Select/Filter/GroupBy

```
1 > etudiants1Df.select("age").show()
2 > etudiants1Df.filter($"age">22).show()
3 > etudiants1Df.groupBy($"departement").count().show()
```

Aggrégation

#### 19.1.2 Sauvegarde

- SaveMod.ErrorIfExists (default)
- SaveMod.Append
- SaveMod.Overwrite
- SaveMod.Ignore

## Save

```

1 > etuDf.write.json("data/etudiants_json")
2 // by default write success if file not exist
3 // use an specific method to write
4 > import org.apache.spark.sql.SaveMode
5 > etuDf.write.mode(SaveMode.Overwrite).json("data/etudiants_json")
6 > etuDf.write.mode(SaveMode.Overwrite).csv("data/etudiants_csv")

```

## 19.1.3 CSV

Par défaut,

## Chargement de CSV

```

1 > import org.apache.spark.sql.types._
2 > val etudiantSchema = StructType(Array(
3   StructField("id", LongType, true),
4   StructField("nom", StringType, true),
5   StructField("age", LongType, true),
6   StructField("departement", StringType, true)))
7 > val etudiants1Df = spark.read.schema(etudiantSchema).option("header", true).csv("etudiants1.csv")
8 > val etudiants2Df = spark.read.schema(etudiantSchema).option("header", true).csv("etudiants2.csv")
9 > val etu2Df = etudiants1Df.union(etudiants2Df)
10 etu2Df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [id: bigint, nom: string ... 2 more fields]
11 > etu2Df.groupBy("departement").sum("age").show()

```

## 19.2 Lab1

## Chargement des fichiers

```

1 > val scanSchema = StructType(Array(
2   StructField("scanId", LongType, true),
3   StructField("productName", StringType, true),
4   StructField("storeId", LongType, true),
5   StructField("sector", StringType, true),
6   StructField("quantity", LongType, true),
7   StructField("totalPrice", FloatType, true)))
8 > val scans = spark.read.schema(scanSchema).option("header", true).csv("scans.csv")
9 > val storeSchema = StructType(Array(
10   StructField("storeId", LongType, true),
11   StructField("postCode", StringType, true),
12   StructField("geoLocation", StringType, true),
13   StructField("name", StringType, true),
14   StructField("town", StringType, true),
15   StructField("type", StringType, true)))
16 > val stores = spark.read.schema(storeSchema).option("header", true).csv("stores.csv")

```

2. enrichir scans avec les données de stores

```

1 > val enrichedScans = scans.join(stores, stores("storeId") === scans("storeId"))

```

3. calculer le chiffre d'affaire

```

1 > enrichedScans.agg(sum("totalPrice").alias("total")).show()

```

4. calculer le chiffre d'affaire par type de magasin

```

1 > enrichedScans.groupBy($"type").sum("totalPrice").show
2 +-----+-----+
3      type sum(totalPrice)
4 +-----+-----+
5 Hypermarche161.19999957084656
6 Supermarche 91.10000014305115
7 +-----+-----+

```

5. calculer le produit ayant la quantité a plus vendu

```

1 > enrichedScans.groupBy("productName").max("quantity").sort($"max(quantity)".desc).first
2 res38: org.apache.spark.sql.Row = [eau,9]
3 // autrement
4 > val maximum = enrichedScans.groupBy("productName")
5   .agg(sum($"quantity").alias("tot_quantity"))
6   .agg(max($"tot_quantity"))
7   .first.toString
8   .substring(1,3).toLong
9 maximum: Long = 24
10 > enrichedScans.groupBy("productName")
11   .agg(sum($"quantity").alias("tot_quantity"))
12   .filter($"tot_quantity"===maximum).show

```

## 19.3 Lab2

Faire de même sous IntelliJ.

build.sbt

```

1 name := "parse-timesheet"
2
3 version := "1.0-SNAPSHOT"
4
5 scalaVersion := "2.11.8"
6
7 libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.1"
8 libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.1.1"

```

Imports

```

1 import org.apache.spark.SparkConf
2 import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}
3 import org.apache.spark.sql.types._
4 import org.apache.spark.sql.functions._

```

Main

```

1 /**
2  * Created by inti on 08/06/17.
3  */
4 object TimesheetDF extends App{
5
6   import TimesheetDFSupport._
7
8   val targetFile = "parse-timesheet-df-result"
9   val conf = new SparkConf().setAppName("parse-timesheet").setMaster("local[*]")
10  val sp = SparkSession.builder()
11    .appName("SparkSession")
12    .master("local[*]")
13    .getOrCreate()
14
15  val timeSheets: DataFrame = groupTimeSheetByDriver(loadsTimeSheets(sp))
16  val drivers: DataFrame = loadDrivers(sp)
17
18  drivers.join(timeSheets, drivers("driverId")===timeSheets("driverIdFromTS"))
19    .select("driverId", "name", "totalHours", "totalMiles")
20    .coalesce(1) // permet de fusionner tous les fichiers en sortie
21    .write.mode(SaveMode.Overwrite).json("data/timeSheetByDrivers_json")
22 }

```

Support

```

1 object TimesheetDFSupport {
2
3   val TimeSheetsFile = "timesheet.csv"
4   val timeSheetSchema = StructType(Array(
5     StructField("driverIdFromTS", LongType, true),
6     StructField("timeSheetId", LongType, true),
7     StructField("hours", LongType, true),

```

```

8   StructField("miles", LongType, true)))
9   val DriversFile = "drivers.csv"
10  val driverSchema = StructType(Array(
11    StructField("driverId", LongType, true),
12    StructField("name", StringType, true),
13    StructField("secu", LongType, true),
14    StructField("zipCode", StringType, true),
15    StructField("town", StringType, true),
16    StructField("wagePlan", StringType, true)))
17
18  def loadsTimeSheets(sp: SparkSession) = {
19    sp.read.schema(timeSheetSchema)
20      .option("header", false)
21      .csv(getClass.getClassLoader.getResource(TimeSheetsFile).getPath)
22  }
23  def loadDrivers(sp: SparkSession) = {
24    sp.read.schema(driverSchema)
25      .option("header", false)
26      .csv(getClass.getClassLoader.getResource(DriversFile).getPath)
27  }
28  def groupTimeSheetByDriver(timeSheet: DataFrame) = {
29    timeSheet.groupBy("driverIdFromTS")
30      .agg(sum("hours").alias("totalHours"), sum("miles").alias("totalMiles"))
31  }
32
33 }

```

## 19.4 Requêtes SQL

La fonction `sql` dans `SparkSession` permet aux applications d'exécuter des requêtes SQL et renvoie le résultat en tant que `DataFrame`. Il est possible d'utiliser les vues temporaires `TempView`.

```

1 > enrichedScans.createOrReplaceTempView("scan")
2 > spark.sql("SELECT * FROM scan").show()
3 > spark.sql("SELECT SUM(totalPrice) FROM scan").show()

```

La `TempView` est déclarée dans une `SparkSession`. Si l'on crée

Support

```

1 > spark.newSession().sql("SELECT SUM(totalPrice) FROM scan").show()
2 org.apache.spark.sql.AnalysisException: Table or view not found: scan; line 1 pos 28
3 > enrichedScans.createGlobalTempView("scan")

```

## 19.5 DataSets

Un `DataSet` est une `DataFrame` généralisée. En effet, une `DataFrame` est un `DataSet` de type `Row`

```
> DataFrame = DataSet[Row]
```

- collection distribuée de données
- nouveau depuis Spark 1.6
- avantages par rapport aux RDD pour utilisation de somme, moyenne, etc. (
- construction à partir d'objet JVM et transformation possible (map, flatMap, filter, ...)
- 
- utilise un encodeur spéciale qui permet à Spark d'effectuer de nombreuses opérations comme le filtrage, le tri et le hachage sans désérialiser les octets dans un objet (n'utilise pas la sérialisation Java ou Kryo)
- les transformations avec les `DataSets` ressemblent au RDD et traitent une type (case class, class,...) au lieu d'une abstraction d'une ligne `Row`.

```

1 > case class Etudiant(age: Long, departement: String, id: Long, nom: String)
2 > val etu1Ds = spark.read.json("etudiants1.json").as[Etudiant]
3 > val etu2Ds = spark.read.json("etudiants2.json").as[Etudiant]

```

```

4 > val etuDs = etu1Ds.union(etu2Ds)
5 |etuDs: org.apache.spark.sql.Dataset[Etudiant] = [age: bigint, departement: string ... 2more fields]|
6 > etuDs.filter(etudiant => etudiant.age < 23).show()

```

En RDD, le compilateur exécutait le code quand on avait une action (i.e. `.collect`). Uniquement à ce moment, on pouvait avoir une exception de notre code. Il en est de même avec les DataFrames, les exceptions sont relevées avec une action (i.e. `.show`).

En DataSet, la compilation est assurée à chaque transformation. Ainsi, les exceptions peuvent être relevées au fur et à mesure du code.

### 19.5.1 Utilisation

Afin de parser un document, il faut avoir créer une `case class` correspond à notre fichier.

- JSON : option `.as[cass class]`

```

1 > case class Etudiant(age: Long, departement: String, id: Long, nom: String)
2 > val etuDs = spark.read.json("etudiants1.json").as[Etudiant]
3 > etuDs.filter(etudiant => etudiant.age < 23).show()

```

- CSV : option `.as[cass class]` + déf `StrutucType` + import `SparkSession.implicit`s

```

1 > case class Etudiant(age: Long, departement: String, id: Long, nom: String)
2 > val etudiantSchema = StructType(Array(
3   StructField("id", LongType, true),
4   StructField("nom", StringType, true),
5   StructField("age", LongType, true),
6   StructField("departement", StringType, true)))
7 > import spark.implicits._
8 > val etu2Ds = spark.read.schema(etudiantSchema)
9   .option("header", true)
10  .csv("etudiants2.csv")
11  .as[Etudiant]

```

NB : Il est nécessaire d'avoir définie une `case class` comportant tous les champs du fichier parsé.

### 19.5.2 Correspondance

`reduceByKey` – `groupByKey` + `reduceGroups`

### 19.5.3 Lab3

#### Imports

```

1 import org.apache.spark.SparkConf
2 import org.apache.spark.sql.{Dataset, SaveMode, SparkSession}
3 import org.apache.spark.sql.types._

```

#### Main

```

1 /**
2  * Created by inti on 08/06/17.
3  */
4 object TimeSheetWithDs extends App{
5
6   import TimeSheetWithDsSupport._
7
8   val targetFile = "parse-timesheet-ds-result"
9
10  val conf = new SparkConf().setAppName("parse-timesheet").setMaster("local[*]")
11  val sp = SparkSession.builder()
12    .appName("SparkSession")
13    .master("local[*]")
14    .getOrCreate()
15
16  val timeSheets: Dataset[TimeSheetFull] = groupTimeSheetByDriver(sp)

```

```

17 val drivers: Dataset[Driver] = convertToDriver(sp)
18
19 import sp.implicitly._
20 drivers.join(timeSheets, drivers("driverId") === timeSheets("driverTsId"))
21   .select("driverId", "name", "hours", "miles")
22   .repartition(2) // indique en combien de fichiers en enregistre les score
23   .coalesce(1) // fusionne en 1 seul fichier
24   .write.mode(SaveMode.Overwrite).json("data/timeSheetByDriversWithDs_json")
25 }

```

## Support

```

1  object TimeSheetWithDsSupport {
2
3      val TimeSheetsFile = "timesheet.csv"
4      val timeSheetSchema = StructType(Array(
5          StructField("driverTsId", LongType, true),
6          StructField("week", LongType, true),
7          StructField("hours", LongType, true),
8          StructField("miles", LongType, true)))
9      val DriversFile = "drivers.csv"
10     val driverSchema = StructType(Array(
11         StructField("driverId", LongType, true),
12         StructField("name", StringType, true),
13         StructField("secu", LongType, true),
14         StructField("zipCode", StringType, true),
15         StructField("town", StringType, true),
16         StructField("wagePlan", StringType, true)))
17
18     def loadsTimeSheets(sp: SparkSession) = {
19         import sp.implicitly._
20         sp.read.schema(timeSheetSchema)
21           .option("header", false)
22           .csv(this.getClass.getClassLoader.getResource(TimeSheetsFile).getPath)
23           .as[TimeSheetFull]
24     }
25     def loadDrivers(sp: SparkSession) = {
26         import sp.implicitly._
27         sp.read.schema(driverSchema)
28           .option("header", false)
29           .csv(this.getClass.getClassLoader.getResource(DriversFile).getPath)
30           .as[DriverFull]
31     }
32     def groupTimeSheetByDriver(sp: SparkSession) = {
33         import sp.implicitly._
34         val timeSheets = loadsTimeSheets(sp)
35         timeSheets.groupByKey(_.driverTsId)
36           .reduceGroups((x,y) => TimeSheetFull(x.driverTsId, x.week, x.hours+y.hours, x.miles+y.miles))
37           .map(_._2)
38     }
39     def convertToDriver(sp: SparkSession) = {
40         import sp.implicitly._
41         loadDrivers(sp).map(el => Driver(el.driverId, el.name))
42     }
43 }

```



## Chapitre 20

# Spark ElasticSearch

### Dépendances

```
1 name := "sparkEs"
2
3 version := "1.0"
4
5 scalaVersion := "2.11.8"
6
7 libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.1"
8 libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.1.1"
9 libraryDependencies += "org.elasticsearch" %% "elasticsearch-spark-20" % "5.4.1"
```

### ajout de la config à SparkContext ou SparkSession

```
1 es.index.auto.create -> true
2 es.nodes -> localhost
3 es.port -> 9200
```

### with RDD

```
1 import org.elasticsearch.spark._
2 rdd.saveToEs("index/type")
```

### with SQL

```
1 import org.elasticsearch.spark.sql._
2 df.saveToEs("index/type")
```

### with Stream

```
1 import org.elasticsearch.spark.streaming._
2 dstream.saveToEs("index/type")
```

Accéder à elasticsearch :

- bin/elasticsearch
- <http://localhost:9200/>
- [http://localhost:9200/\\_plugin/head](http://localhost:9200/_plugin/head)

### Boot

```
1 object BootWithDs extends App {
2   import DsSupport._
3   val conf = new SparkConf().setMaster("local[*]")
4     .setAppName("SparkEs")
5     .set("es.index.auto.create", "true")
6     .set("es.nodes", "localhost")
7     .set("es.port", "9200")
8   val sp = SparkSession.builder()
9     .master("local[*]")
10    .appName("SparkSessionEs")
11    .getOrCreate()
```

```

12 val timeSheets: Dataset[TimeSheetFull] = groupTimeSheetByDriver(sp)
13 val drivers: Dataset[Driver] = convertToDriver(sp)
14
15 import sp.implicit._
16 drivers.join(timeSheets, drivers("driverId") === timeSheets("driverTsId"))
17   .select("driverId", "name", "hours", "miles")
18   .saveToEs("driver/driverStats")
19 }

```

## 20.1 Lab

### Kafka Producer

```

1 import java.util.Properties
2 import org.apache.kafka.clients.producer._
3 import scala.io.Source
4 import scala.util.Random
5 /**
6  * Created by inti on 09/06/17.
7  */
8 object ScanPoducer extends App {
9
10  val topic = "scanstream"
11  val props = new Properties()
12  props.put("bootstrap.servers", "localhost:9092")
13  props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")
14  props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")
15  val producer = new KafkaProducer[String, String](props)
16  println("Start producing ...")
17  while(true) {
18    Source.fromFile(getClass.getClassLoader.getResource("scans.csv").getPath)
19      .getLines()
20      .foreach(line => {
21        val record = new ProducerRecord(topic, "key", line)
22        Thread.sleep(1 + Random.nextInt(100))
23        producer.send(record)
24      })
25  }
26  producer.close()
27  println("finish")
28 }
29

```

### Commande REST pour récupérer le chiffre d'affaire en temps réel

```

1 POST scans/scan/_search
2 {
3   "size": 0,
4   "aggs" : {
5     "sum_totalPrice" : { "sum" : { "field" : "totalPrice" } }
6   }
7 }

```

## 20.2 Kibana

- framework d'ElasticSearch
- Open config/kibana.yml in an editor and set `elasticsearch.url` to point at your Elasticsearch instance
- run `bin/kibana`
- <http://localhost:5601>
- choisir l'index d'ES et une **key** correspondant à une date.

**NB :** Kibana a besoin d'une date d'événements dans les index sur lesquels il travaille.

case class parsant les données

```
1 import java.util.Date
2 case class ScanWithDate(
3     scanId: Long,
4     productName: String,
5     storeId: Long,
6     sector: String,
7     quantity: Long,
8     totalPrice: Float,
9     eventDate: Date
10 )
```

- création de visualisations : Histogram, Pie Chart, ...
- création de dashboard

## 20.3 Quiz

1. Pourquoi utiliser Spark ? Citer des exemples
2. Spark est un framework MapReduce In-Memory
3. Comment les transformations sont déclenchés
4. Spark Batch VS Spark Streaming
5. Dataset VS DataFrame
6. Décrire ces exécutions sur un cluster de 3 noeuds : 1 driver et 2 exécutateurs

```
1 val data = Array(1,2,1,2,1,3,8)
2 val distData = sc.parallelize(data)
3 val result = sc.parallelize(data)
4     .filter(x => x<3)
5     .map(x => (x,x))
6     .reduceByKey((x,y) => x+y)
7     .collect()
```

## Chapitre 21

### D3JS

# Chapitre 22

## Projet

### 22.1 Twitter Streaming

sbt dependencies

```
1 libraryDependencies += "org.apache.spark" % "spark-streaming_2.11" % "2.1.1"
2 libraryDependencies += "org.apache.bahir" %% "spark-streaming-twitter" % "2.1.0"
```

### 22.2 Twitter Search

sbt dependencies

```
1 libraryDependencies += "org.apache.spark" % "spark-streaming_2.11" % "2.1.1"
2 libraryDependencies += "org.apache.bahir" %% "spark-streaming-twitter" % "2.1.0"
```

### 22.3 Spark ML

sbt dependencies

```
1 libraryDependencies += "org.apache.spark" % "spark-mllib_2.11" % "2.1.1"
```

See <https://github.com/examples> See Multilingualsentiment

See Dashboard

# Bibliographie

- [1] M. NEBRA, *Réaliser votre site web avec HTML5 et CSS3*. Le site du Zero, 2005 (cf. p. 35).
- [2] S. DE LA MARK et J. PARDANAUD, *Dynamiser vos sites web avec JavaScript*. Le site du Zero, 2005 (cf. p. 36).
- [3] M. MARTIN, *Simplifiez vos développements JavaScript avec JQuery*. Le site du Zero, 2005 (cf. p. 37).
- [4] M. TOM, *Machine Learning*. McGraw Hill, 1997 (cf. p. 95).
- [5] A. ETHEM, *Introduction to Machine Learning*. MIT Press, 2010 (cf. p. 95).