

Document Data Pipeline

Exemple avec Web scraping, MongoDB et agrégation

Xavier Gendre

Introduction

Un pipeline de traitement de données (*data pipeline*) fait le lien entre des producteurs de données (*data producers*) et des consommateurs de données (*data consumers*).

Objectif : présenter un exemple simple de pipeline de traitement de données orientée document.

- Récupération de données depuis des sites internet
- Filtrage et mise en forme des données
- Stockage dans une base de données
- Exploitation et valorisation des données

JSON, un format document

Notion de document

Pas de réelle définition mais quelques propriétés :

- un **document** contient des données encodées dans un certain **format**,
- le **contenu** d'un document est un ensemble de paires **clé/valeur**,
- absence de **schéma** fixe a priori,
- un document peut contenir d'autres documents.

Exemples :

- format texte : **JSON**, XML, YML, ...
- format binaire : **BSON**, PDF, ...

Format JSON

JSON (*JavaScript Object Notation*) est un format document léger et très utilisé qui se veut lisible autant par des humains que par des machines.

La syntaxe est simple et un document se présente comme un **objet unique** délimité par des accolades `{}`. Les paires clé/valeur sont séparées par des virgules.

```
{  
    "clé1": valeur1,  
    "clé2": valeur2,  
    ...  
}
```

Les clés sont des chaînes de caractères délimitées par des guillemets `" "` et tous les caractères blancs (espace, tabulation et retour à la ligne) sont ignorés.

Format JSON - Types simples

- **chaîne de caractères** : séquence de caractères Unicode délimitée par des guillemets "",
- **nombre** : flottants double précision (notation entière, décimale ou scientifique),
- **booléen** : true et false (en minuscules),
- **null** : valeur vide.

```
{  
  "nom": "Gandalf",  
  "taille": 1.68,  
  "anneaux": 1,  
  "magicien": true,  
  "résidence": null  
}
```

Format JSON - Types structurés

- **tableau** : liste ordonnée de valeurs entre crochets [] (similaire aux vecteurs de R),
- **objet** : liste non ordonnée de paires clé/valeur entre accolades {} (similaire aux listes de R).

```
{  
  "nom": "Gandalf",  
  "stuff": ["Glamdring", "Bâton de magicien", "Narya"],  
  "alias": {  
    "sindarin": "Mithrandir",  
    "quenya": "Olórin"  
  }  
}
```

Format JSON - Imbrication

Pour un type structuré, les valeurs peuvent être de n'importe quel type y compris un autre type structuré.

```
{  
  "nom": "Sacquet", "prénom": "Frodon",  
  "amis": [  
    {  
      "nom": "Gamegie", "prénom": "Samsagace"  
    },  
    {  
      "nom": "Brandebouc", "prénom": "Meriadoc"  
    },  
    {  
      "nom": "Touque", "prénom": "Peregrin"  
    }  
  ]  
}
```

Package jsonlite

Le package jsonlite permet de manipuler simplement des données document au format JSON avec R.

```
library(jsonlite)
json <- toJSON(list(nom="Gandalf", taille=1.68, residence=NA), pretty=TRUE)
typeof(json); print(json)

## [1] "character"

## {
##   "nom": ["Gandalf"],
##   "taille": [1.68],
##   "residence": [null]
## }
```

Package **jsonlite** - Import/Export

La fonction **toJSON** permet d'exporter un objet R en JSON et la fonction **fromJSON** permet d'importer un objet JSON dans R.

```
df <- fromJSON(json)
typeof(df); print(df)
```

```
## [1] "list"
```

```
## $nom
## [1] "Gandalf"
##
## $taille
## [1] 1.68
##
## $residence
## [1] NA
```

Package **jsonlite** - Objets JSON et R

Le package **jsonlite** met (presque) en place une bijection entre la structure des données dans un document JSON et l'objet R retourné par **fromJSON**.

```
obj <- list(nom="Sauron", residence="Mordor")  
toJSON(obj)
```

```
## {"nom": [ "Sauron" ], "residence": [ "Mordor" ]}
```

```
all.equal(obj, fromJSON(toJSON(obj)))
```

```
## [1] TRUE
```

Package `jsonlite` - Data frames

```
obj <- data.frame(  
  nom=c("Aragorn", "Boromir"),  
  age=c(210, 41),  
  maison=c("Isildur", "Húrin"),  
  stringsAsFactors=FALSE)  
  
all.equal(obj,  
         fromJSON(toJSON(obj)))  
  
## [1] TRUE  
  
toJSON(obj, pretty=TRUE)  
  
## [1]  
## {  
##   "nom": "Aragorn",  
##   "age": 210,  
##   "maison": "Isildur"  
## },  
## {  
##   "nom": "Boromir",  
##   "age": 41,  
##   "maison": "Húrin"  
## }  
## ]
```

Package `jsonlite` - Quelques remarques

- Les valeurs manquantes ne sont tout simplement pas stockées.

```
toJSON(data.frame(v1=c(FALSE, TRUE, NA, NA),  
                  v2=c("Elfe", NA, NA, "Tolkien")))
```

```
## [{"v1":false,"v2":"Elfe"}, {"v1":true}, {}, {"v2":"Tolkien"}]
```

- Pour gérer des **flux** de données, le package `jsonlite` propose les fonctions `stream_in` et `stream_out`. Celles-ci permettent de manipuler des données au format NDJSON, i.e. du JSON “ligne par ligne”. Nous y reviendrons ...

```
stream_out(obj) # stream_out(obj, file("/tmp/dump.json")) pour exporter dans un fichier
```

```
## {"nom": "Aragorn", "age": 210, "maison": "Isildur"}  
## {"nom": "Boromir", "age": 41, "maison": "Húrin"}  
##  
Complete! Processed total of 2 rows.
```

Obtenir du JSON avec une API

De nombreux sites proposent des API pour faire des requêtes et retournent les résultats au format JSON.

Quelques exemples :

[<https://lotr.fandom.com/api/v1/Articles/Details?ids=61&abstract=100>](https://lotr.fandom.com/api/v1/Articles>List/?limit=25</p></div><div data-bbox=)

<https://lotr.fandom.com/api/v1/Search>List/?query=bombadil&limit=10&minArticleQuality=95&batch=5>

Ces interfaces peuvent être utilisées dans un navigateur (peu utile), en ligne de commande (cURL, ...) et, bien sûr, avec R.

Obtenir du JSON avec une API

```
articles <- fromJSON("https://lotr.fandom.com/api/v1/Articles>List/?limit=25")
names(articles)

## [1] "items"    "basepath"  "offset"

articles$items[1,]

##      id                      title
## 1 17037 "I Can't Carry It for You... but I Can Carry You."
##                                         url ns
## 1 /wiki/%22I_Can%27t_Carry_It_for_You..._but_I_Can_Carry_You.%22  0

nrow(articles$items)

## [1] 25
```

Regrouper des données

La plupart du temps, les API limitent le nombre de réponses par requête et imposent l'utilisation de pages qui peuvent être combinées dans un data frame avec `rbind_pages`.

```
url <- "https://lotr.fandom.com/api/v1/Search>List/?query=bombadil&limit=10"
pages <- list()
for (i in 1:5) {
  data_page <- paste0(url, "&batch=", i) %>% fromJSON()
  pages[[i]] <- data_page$items
}
items <- rbind_pages(pages)
nrow(items)

## [1] 50
```

À vous de jouer!

Web scraping

Le Web sans API

Les sites Web sont une source importante d'information :

- site interne d'entreprise,
- Wikipedia,
- IMDb,
- ...

Tous les sites ne disposent pas d'une API pour récolter de l'information (outil non pertinent, manque de moyens, modèle économique, ...).

Cependant, l'information est assez structurée pour que nous puissions la récupérer de façon semi-automatisée.

Objectif

Cate Blanchett interprète le rôle de *Galadriel* dans les films de Peter Jackson. Nous allons chercher avec quel acteur Cate Blanchett a le plus joué dans les années 2000.

Nous pouvons commencer par visiter la section dédiée à la filmographie de Cate Blanchett sur sa page Wikipedia.

https://fr.wikipedia.org/wiki/Cate_Blanchett#Filmographie

Nous constatons que :

- les films sont classés par décennie,
- chaque titre de film (sauf un) possède un lien vers une page dédiée,
- ces pages contiennent des listes d'acteurs.

Package `rvest`

Le package `rvest` permet de manipuler le contenu de pages Web depuis R en travaillant sur son contenu brut (**HTML** et **CSS**). Son installation demande quelques prérequis extérieurs à R tels que le programme **cURL** mais tout est expliqué à l'installation.

```
install.packages("rvest")
```

Pour utiliser `rvest`, cela commence comme avec n'importe quel autre package.

```
library(rvest)
```

```
## Loading required package: xml2
```

HTML et CSS

Le travail d'un navigateur Web est de transformer des fichiers textes en pages lisibles et bien présentées. En simplifiant le fonctionnement, cela se décompose dans :

- des **informations structurées** contenues dans un fichier HTML,
- du **code de présentation** dans des feuilles de style CSS.

Ainsi, c'est dans le fichier source HTML que nous allons trouver les informations que nous voulons.

Arbre DOM

Le contenu d'un fichier HTML est un format document comme ce que nous avons vu pour le JSON.

Les données sont organisées de façon **arborescente**.

```
<h4>Longs-métrages</h4>
  <h5>Années 1990</h5>
  <ul>
    <li>...</li>
  </ul>
```

L'interface de programmation **Document Object Model (DOM)** permet d'examiner et de modifier cet arbre et donc le contenu d'une page Web. C'est de cette façon que des pages Web dynamiques sont créées.

Premier essai - Données

Le package **rvest** permet de parcourir l'arbre DOM et de récupérer des informations.

```
url_wikipedia <- "https://fr.wikipedia.org/"  
url_blanchett <- "wiki/Cate_Blanchett"  
url <- paste0(url_wikipedia, url_blanchett)  
  
data_html <- read_html(url)  
data_html  
  
## {html_document}  
## <html class="client-nojs" lang="fr" dir="ltr">  
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...  
## [2] <body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-subject" ...
```

Premier essai - Titres de section

Les éléments de l'arbre sont appelés des **nœuds** et peuvent être extraits avec la fonction `html_nodes`. Dans notre cas, nous sommes intéressés par les sections de niveau h5.

```
data_html %>% html_nodes("h5") %>% head(1)

## {xml_nodeset (1)}
## [1] <h5>\n<span id="Ann.C3.A9es_1990"></span><span class="mw-headline" id="An ...
```

`html_nodes` donne plus d'informations que le seul contenu textuel du titre.

```
data_html %>% html_nodes("h5") %>% head(1) %>% html_text()

## [1] "Années 1990[modifier | modifier le code]"
```

Sélectionner des nœuds

La fonction `html_nodes` joue un rôle central pour le web scraping car elle permet de sélectionner les nœuds contenant les informations d'intérêt. Pour faire cette sélection, deux méthodes sont disponibles :

- **sélecteurs CSS** : motifs CSS pour identifier une catégorie d'éléments (`p` pour les paragraphes, `h5` pour les titres de niveau 5, ...),
- **XPath** : standard W3C pour identifier des informations dans un document XML.

Les sélecteurs CSS sont plus simples et conviennent pour une utilisation occasionnelle. Pour un usage plus fréquent, la puissance de XPath sera privilégiée.

Concrètement, le package `rvest` n'utilise que XPath et les sélecteurs CSS sont silencieusement transformés en syntaxe XPath.

Sélectionner des nœuds - Sélecteurs CSS

Voici comment récupérer les attributs (fonction `html_attrs`) des nœuds des films de Cate Blanchett des années 2000.

```
css_select <- '#mw-content-text > div > ul:nth-of-type(3) > li > i:nth-of-type(1) > a'  
films_css <- data_html %>% html_nodes(css_select) %>% html_attrs()  
length(films_css)  
  
## [1] 24  
  
films_css[[1]]  
  
## href title  
## "/wiki/Les_Larmes_d'un_homme" "Les Larmes d'un homme"
```

Sélectionner des nœuds - Sélecteurs CSS

Voici comment récupérer les attributs (fonction `html_attrs`) des nœuds des films de Cate Blanchett des années 2000.

```
css_select <- '#mw-content-text > div > ul:nth-of-type(3) > li > i:nth-of-type(1) > a'
```

Quelques explications :

- `#mw-content-text` est un sélecteur d'identifiant (`id="..."` en CSS),
- le sélecteur d'enfant, noté `>`, permet d'utiliser la filiation pour désigner un élément par rapport à un autre,
- `ul:nth-of-type(3)` signifie que nous ne voulons extraire que le 3ème élément `ul` dans la liste des enfants (pseudo-classe CSS),

Mais... D'où est-ce que ça sort ? (Réponse brutale : Copy > CSS Selector mais nous venons de voir comment être plus fins)

Sélectionner des nœuds - XPath

Pour utiliser XPath, il faut utiliser explicitement l'argument `xpath`.

```
xpath_str <- '//*[@id="mw-content-text"]  
/div/ul[  
  preceding::h5[span/@id="Années_2000"]  
  and  
  following::h5[span/@id="Années_2010"]  
]/li/i[1]/a'  
films_xpath <- data_html %>% html_nodes(xpath=xpath_str) %>% html_text()  
length(films_xpath)  
  
## [1] 24  
  
films_xpath %>% head(3)  
  
## [1] "Les Larmes d'un homme"          "Intuitions"  
## [3] "Bandits : Gentlemen braqueurs"
```

Sélectionner des nœuds - XPath

Pour utiliser XPath, il faut utiliser explicitement l'argument `xpath`.

```
xpath_str <- '//*[@id="mw-content-text"]  
/div/ul[  
  preceding::h5[span/@id="Années_2000"]  
  and  
  following::h5[span/@id="Années_2010"]  
]/li/i[1]/a'
```

Quelques explications :

- `//*[@id="mw-content-text"]` sélectionne le sous-arbre de tous les nœuds descendants de cet identifiant,
- `/ul[...]` filtre les `` dans ce sous-arbre qui suivent une balise `<h5>` contenant un `` identifié par `Années_2000` et précèdent une balise `<h5>` contenant un `` identifié par `Années_2010`,
- `/i[1]` sélectionne le premier `<i>` de la liste.

Distribution d'un film

Appliquons ce que nous venons de voir pour récupérer la liste des acteurs du film *Heaven* (2002).

```
url_film <- "wiki/Heaven_(film,_2002)"
url <- paste0(url_wikipedia, url_film)
data_html <- read_html(url)
data_html

## {html_document}
## <html class="client-nojs" lang="fr" dir="ltr">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-subject" ...
```

Le découpage entre `url_wikipedia` et `url_film` sera important pour automatiser cette démarche.

Distribution d'un film

```
xpath_str <- '(
  //*[@id="mw-content-text"]//ul[preceding::h2[span/@id="Distribution"]]
)[1]/li/a[1]'

data_html %>% html_nodes(xpath=xpath_str) %>% html_text()

## [1] "Cate Blanchett"      "Giovanni Ribisi"      "Remo Girone"
## [4] "Stefania Rocca"      "Mattia Sbragia"      "Stefano Santospago"
## [7] "Alberto Di Stasio"   "Giovanni Vettorazzo" "Gianfranco Barra"
## [10] "Vincenzo Ricotta"    "Mauro Marino"
```

Quelques explications :

- les parenthèses () permettent de définir un sélecteur complexe,
- un seul slash / sélectionne **depuis le nœud courant** alors qu'un double // sélectionne **tous les descendants** du nœud courant.

Application

Nous commençons par la liste des films.

```
url_wikipedia <- "https://fr.wikipedia.org/"  
url_blanchett <- "wiki/Cate_Blanchett"  
data_html <- paste0(url_wikipedia, url_blanchett) %>% read_html()  
  
xpath_films <- '//*[@id="mw-content-text"]'  
  /div/ul[  
    preceding::h5[span/@id="Années_2000"]  
    and  
    following::h5[span/@id="Années_2010"]  
  ]/li/i[1]/a'  
films <- data_html %>% html_nodes(xpath=xpath_films) %>% html_attrs()  
  
length(films)  
  
## [1] 24
```

Application

Chaque film a un titre (**title**) et un lien vers sa page (**href**) ...

```
films[[1]]
```

```
## href title
## "/wiki/Les_Larmes_d'un_homme" "Les Larmes d'un homme"
```

... sauf un!

```
films[[15]]
```

```
## href
## "/w/index.php?title=Stories_of_Lost_Souls&action=edit&redlink=1"
## class
## "new"
## title
## "Stories of Lost Souls (page inexisteante)"
```

Application

Nous pouvons maintenant récolter la liste des acteurs.

```
acteurs <- tibble()
xpath_dist <- '(
  //*[@id="mw-content-text"]//ul[preceding::h2[span/@id="Distribution"]]
)[1]/li/a[1]'

for(i in seq_along(films)) {
  if("class" %in% names(films[[i]])) next # Absence de page dédiée
  url_film <- films[[i]][["href"]]
  data_html <- paste0(url_wikipedia, url_film) %>% read_html()
  film_dist <- data_html %>% html_nodes(xpath=xpath_dist) %>% html_text()
  acteurs <- acteurs %>% rbind(tibble(nom=film_dist, titre=films[[i]][["title"]]))
}
```

Application

La réponse s'obtient grâce aux outils de `dplyr`.

```
acteurs %>%  
  group_by(nom) %>%  
  summarise(n=n(), .groups='drop') %>%  
  arrange(desc(n)) %>%  
  head(4)  
  
## # A tibble: 4 x 2  
##   nom                 n  
##   <chr>              <int>  
## 1 Cate Blanchett     20  
## 2 Hugo Weaving       4  
## 3 John Rhys-Davies   4  
## 4 Billy Boyd          3
```

À égalité, il s'agit de Hugo Weaving (*Elrond*) et de John Rhys-Davies (*Gimli*) !

À vous de jouer!

MongoDB, une base de données
orientée documents

Présentation

Il s'agit d'un **logiciel libre** sous double licences (GNU AGPL v3.0 et Apache v2.0).

La communication se fait selon le principe **client-serveur**. Il s'agit d'un système de gestion de base de données populaire et réputé facile d'utilisation :
<http://db-engines.com/en/ranking>

La langue maternelle de MongoDB est le **JavaScript** pour l'exécution de fonctions côté serveur. Les objets manipulés sont au format **BSON** (JSON binaire).



NoSQL

MongoDB fait partie de la mouvance NoSQL au titre de système de gestion de base de données **orienté documents**.

Les données manipulées sont des documents enrichis d'un champ `_id` et enregistrés dans des **collections**.

- **Imbrication de documents** : cela permet une **approche relationnelle**.
- **Absence de schéma** : pour créer un document (ou une collection), il suffit de l'utiliser mais cette simplicité implique que **toute faute de frappe peut avoir des conséquences importantes**.

Package `mongolite`

Le package `mongolite` permet de se connecter à un serveur MongoDB local ou distant et d'interagir avec lui depuis R. Son installation demande que les bibliothèques de développement de `OpenSSL` et `Cyrus SASL` soient présentes sur le système mais tout est expliqué à l'installation.

```
install.packages("mongolite")
```

Nous pouvons ensuite charger le package pour commencer à travailler avec MongoDB depuis R.

```
library(mongolite)
```

Connexion

La première étape est de se connecter au serveur MongoDB. Par défaut, la connexion est locale (**localhost**) mais elle peut être distante.

```
user <- "readwrite"
pass <- "test"
host <- "mongo.opencpu.org"
port <- "43942"
path <- "/jeroen_test"
opts <- "?retryWrites=false"
url <- paste0("mongodb://", user, ":", pass, "@", host, ":", port, path, opts)
```

Pour ouvrir la connexion avec le serveur MongoDB, nous utilisons la fonction **mongo** avec un nom de collection pour récupérer un objet R permettant d'interagir avec elle.

```
collection <- "change_me" # Choisir un nom pour votre collection
m <- mongo(collection, url=url) # mongo(collection) pour une connexion locale
```

Premier contact

La collection n'a pas besoin d'être préalablement créée et l'objet retourné par `mongo` offre plusieurs méthodes pour la manipuler. Par exemple, `count` sans paramètre retourne le nombre de documents.

```
m$count()
```

```
## [1] 999
```

Si la collection a déjà été utilisée, elle peut être vidée avec la méthode `drop`.

```
if(m$count() > 0) m$drop()
```

Attention, sur un serveur partagé, la collection peut être en cours d'utilisation par une autre personne! Il est conseillé de choisir un nom original pour faire des essais sans être perturbé.

Insertion

La méthode `insert` permet d'ajouter des éléments à la collection.

```
# Cartes du jeu "The Lord of the Rings: The Card Game"  
m$insert( fromJSON("https://ringsdb.com/api/public/cards" ) )  
  
## List of 5  
## $ nInserted : num 999  
## $ nMatched : num 0  
## $ nRemoved : num 0  
## $ nUpserted : num 0  
## $ writeErrors: list()  
  
m$count()  
  
## [1] 999
```

Insertion

L'absence de schéma permet d'insérer des documents qui n'ont pas la même structure.

```
m$insert(list(name="Luke Skywalker", outlier=TRUE))
```

```
## List of 6
## $ nInserted : int 1
## $ nMatched  : int 0
## $ nModified : int 0
## $ nRemoved  : int 0
## $ nUpserted : int 0
## $ writeErrors: list()
```

```
m$count()
```

```
## [1] 1000
```

Recherche

La méthode **find** permet de faire une recherche dans la collection. Sans paramètre, toute la collection est retournée.

```
m$find()
```

Il est possible de définir des critères de recherche avec l'argument **query**. Le format JSON est utilisé pour cela.

```
m$find(query='{"type_name": "Contract"})'
```

L'argument **fields** permet de limiter les champs retournés (par défaut, **_id** est toujours retourné).

```
m$find(query='{"type_name": "Contract"})', fields='{"name": 1, "illustrator": 1}'')
```

Recherche - Exemple

```
m$find(query='{"type_name": "Contract"}',  
       fields='{"_id": 0, "pack_name": 1, "name": 1, "illustrator": 1}')
```

| | pack_name | name | illustrator |
|------|-----------------------------|---------------------------|----------------|
| ## 1 | A Shadow in the East | Fellowship | Leanna Crossan |
| ## 2 | Wrath and Ruin | The Burglar's Turn | Greg Bobrowski |
| ## 3 | The City of Ulfast | Forth, The Three Hunters! | Justin Gerard |
| ## 4 | Challenge of the Wainriders | The Grey Wanderer | Justin Gerard |
| ## 5 | Under the Ash Mountains | Council of the Wise | Borja Pindado |
| ## 6 | The Land of Sorrow | Messenger of the King | Justin Gerard |
| ## 7 | The Fortress of Nurn | Bond of Friendship | Borja Pindado |

Recherche - Opérateurs

Il est possible d'utiliser des opérateurs dans l'argument `query`. Ils sont précédées du caractère '\$'.

- `$exists` pour tester l'existence d'un champ,

```
m$find(query='{"outlier": {"$exists": true} }', fields='{"_id": 0, "name": 1}')
```

```
##           name
## 1 Luke Skywalker
```

- `$and`, `$or`, `$nor` et `$not` pour la logique,

```
m$find(query='{"$and": [{"type_name": "Contract"}, {"illustrator": "Leanna Crossan"}]}',
       fields='{"_id": 0, "name": 1}')
```

```
##           name
## 1 Fellowship
```

Recherche - Opérateurs

- `$regex` pour utiliser des expressions régulières,

```
m$find(query='{"name": {"$regex": "^Z", "$options" : "i"} }',  
       fields='{"_id": 0, "name": 1}')
```

```
##           name  
## 1 Zigel Miner
```

- `$lt`, `$lte`, `$gt` et `$gte` pour comparer des nombres,

```
m$find(query='{"attack": {"$gte": 4} }',  
       fields='{"_id": 0, "name": 1, "attack": 1}') %>% head(3)
```

```
##           name attack  
## 1   Gandalf      4  
## 2   Saruman      5  
## 3 Treebeard      4
```

Recherche - Opérateurs

- `$ne` pour tester la non-égalité,

```
m$find(query='{"pack_name": {"$ne": "Core Set"} }',  
       fields='{"_id": 0, "pack_name": 1, "name": 1}') %>% head(3)
```

```
##          pack_name      name  
## 1 The Hunt for Gollum  Bilbo Baggins  
## 2 The Hunt for Gollum  Dúnedain Mark  
## 3 The Hunt for Gollum Campfire Tales
```

- Et bien d'autres : `$in` pour l'inclusion, `$nin` pour la non-inclusion, `$all` pour l'inclusion stricte, `$size` pour la taille d'un tableau, `$type` pour le type d'une valeur, `$mod` pour le modulo, ...

<https://docs.mongodb.com/manual/reference/operator/query/>

Compter

La méthode `count` permet de compter les documents qui vérifient des conditions données.

```
m$count('{"pack_name": "Core Set"}')  
## [1] 73  
  
m$count('{$and': [  
  {"pack_name": "Core Set"},  
  {"type_name": {"$in": ["Hero", "Contract"]}}]  
}')  
## [1] 12
```

Valeurs uniques

La méthode `distinct` retourne la liste des valeurs prises par une clé parmi les documents vérifiant une recherche donnée.

```
m$distinct(key="type_name")
```

```
## [1] "Hero"          "Ally"          "Event"  
## [4] "Attachment"   "Player Side Quest" "Treasure"  
## [7] "Contract"
```

```
m$distinct(key="illustrator", query='{"pack_name": "The Hunt for Gollum"}')
```

```
## [1] "Tony Foti"      "Joko Mulyono"    "Felicia Cano"  
## [4] "David A. Nash" "Jake Murray"     "Magali Villeneuve"  
## [7] "Stu Barnes"     "Katherine Dinger" "Ben Zweifel"  
## [10] "John Gravato"
```

Tri

L'argument `sort` de la méthode `find` permet de trier les résultats selon les valeurs d'un champ (1 pour l'ordre croissant et -1 pour le décroissant).

```
m$find(  
  query='{"threat": {"$exists": true} }',      # Supprime les NA  
  fields='{"_id": 0, "name": 1, "threat": 1}', # Le nom et le niveau de menace  
  sort='{"threat": -1}'                      # En partant du plus menaçant  
) %>% head(5)  
  
##          name threat  
## 1    Gandalf     14  
## 2    Saruman     13  
## 3    Gwaihir      13  
## 4 (MotK) Beorn     13  
## 5    Elrond       13
```

Suppression

Nous avons déjà vu `drop` pour vider une collection. La méthode `remove` supprime les entrées correspondant à une recherche donnée.

```
m$count()  
## [1] 1000  
  
m$remove(query='{"outlier": {"$exists": true} }')  
m$count()  
## [1] 999
```

Itération

La méthode `iterate` prend les mêmes arguments que `find` et retourne un itérateur pour parcourir le résultat de la recherche ligne par ligne à l'aide de `$one` (ou par page avec `$page`). Une fois le dernier élément retourné, l'itérateur devient obsolète.

```
it <- m$iterate(query='{"$and": [{"threat": {"$gte": 13} },
                                {"health": {"$gte": 5} }]}',
                 fields='{"_id": 0, "name": 1}',
                 sort='{"name": 1}')
while(!is.null(item <- it$one())) {
  cat("Respect à toi, ô", item$name, "!\n")
}

## Respect à toi, ô (MotK) Beorn !
## Respect à toi, ô Gandalf !
## Respect à toi, ô Treebeard !
```

Import/Export

Le format par défaut pour ces opérations est le NDJSON (*Newline Delimited JSON*) que nous avons évoqué pour la gestion des flux avec `jsonlite`.

Les méthodes `export` et `import` sont très simples.

```
# Exporte la collection vers un fichier
m$export(file("/tmp/dump.json"))
# Purge la collection
m$drop(); m$count()
```

```
## [1] 0
```

```
# Importe la collection depuis un fichier
m$import(file("/tmp/dump.json"))
m$count()
```

```
## [1] 999
```

Import/Export - Gestion des flux

Grâce aux fonctions `stream_in` et `stream_out` de `jsonlite`, nous pouvons mettre en place des mécanismes élégants pour importer des données obtenues par flux (réseau, capteur, ...).

Un flux se décompose en **pages** de taille donnée et le principe est de stocker chaque page (fichier temporaire, mémoire, ...) avant de l'importer dans la collection.

L'avantage de cette approche est que nous pouvons faire agir une fonction `handler` à ce moment pour manipuler les données avant l'import.

Import/Export - Gestion des flux (Exemple)

La variable `traits` des données de *The Lord of the Rings: The Card Game* est une chaîne de caractères qui contient plusieurs informations. Ce n'est pas bien formaté et nous souhaitons la remplacer par un tableau contenant ces informations pour tester l'inclusion (opérateur `$in`).

```
m$find(fields='{"_id": 0, "name": 1, "traits": 1}') %>% head(2)
```

```
##          name           traits
## 1 Aragorn Dúnedain. Noble. Ranger.
## 2 Théodred   Noble. Rohan. Warrior.
```

```
m$find(query='{"traits": {"$in": ["Dwarf"]}} ',
       fields='{"_id": 0, "name": 1, "traits": 1}')
```

```
## data frame with 0 columns and 0 rows
```

Import/Export - Gestion des flux (Exemple)

```
# Purge la collection
m$drop()

# Fichier temporaire pour stocker les pages
ftmp <- file(tempfile(), open="w+b")
# Importation d'un flux depuis un fichier (400 documents par page)
file("/tmp/dump.json") %>%
  stream_in(pagesize=400,           # Taille des pages
            handler=function(df) { # Handler de page
              cat(" [", nrow(df), "ligne(s) lue(s) ]")
              df$traits <- df$traits %>%
                strsplit(split=' ') %>%
                sapply(function(s) { gsub('\\.', ' ', s) })
              stream_out(df, ftmp)
              m$import(ftmp)
            })
# Fermeture du fichier temporaire
close(ftmp)
```

Import/Export - Gestion des flux (Exemple)

```
## using a custom handler function.

## opening file input connection.

## [ 400 ligne(s) lue(s) ]
Complete! Processed total of 400 rows.
##
  Found 400 records... [ 400 ligne(s) lue(s) ]
Complete! Processed total of 400 rows.
##
  Found 800 records... [ 199 ligne(s) lue(s) ]
Complete! Processed total of 199 rows.
##
  Found 999 records...

## closing file input connection.
```

Import/Export - Gestion des flux (Exemple)

Il est possible de rendre la sortie moins verbeuse avec l'option `verbose=FALSE`.

Voici le résultat :

```
m$find(fields='{"_id": 0, "name": 1, "traits": 1}') %>% head(2)
```

```
##           name          traits
## 1 Aragorn Dúnedain, Noble, Ranger
## 2 Théodred    Noble, Rohan, Warrior
```

```
m$find(query='{"traits": {"$in": ["Dwarf"]}} ',
       fields='{"_id": 0, "name": 1, "traits": 1}') %>% head(2)
```

```
##           name          traits
## 1 Glóin        Dwarf, Noble
## 2 Gimli Dwarf, Noble, Warrior
```

Modification

Pour modifier un document de la collection, la méthode `update` procède en deux temps :

- rechercher le(s) document(s) à modifier,
- apporter les modifications au(x) document(s).

Par défaut, un seul document peut être modifier. Pour appliquer les modifications à plusieurs documents, il faut utiliser l'argument `multiple=TRUE`.

La recherche se fait avec l'argument `query` et les modifications avec `update`.

Modification - Changer une valeur

Utilisons `$set` pour changer la valeur d'une clé (ou créer la paire clé/valeur).

```
m$update(query='{"name": "Gandalf"}',  
         update='{"$set": {"cheveux": "gris"}}')  
  
m$find(query='{"name": "Gandalf"}', fields='{"_id": 0, "name": 1, "cheveux": 1}')  
  
##      name cheveux  
## 1 Gandalf    gris  
## 2 Gandalf    <NA>  
## 3 Gandalf    <NA>  
## 4 Gandalf    <NA>  
## 5 Gandalf    <NA>
```

Modification - Changer une valeur

Utilisons `$set` pour changer la valeur d'une clé (ou créer la paire clé/valeur).

```
m$update(query='{"name": "Gandalf"}',
          update='{"$set": {"cheveux": "gris"}}',
          multiple=TRUE) # Tous les documents sont modifiés

m$find(query='{"name": "Gandalf"}', fields='{"_id": 0, "name": 1, "cheveux": 1}')

##      name cheveux
## 1 Gandalf    gris
## 2 Gandalf    gris
## 3 Gandalf    gris
## 4 Gandalf    gris
## 5 Gandalf    gris
```

Modification - Supprimer une paire clé/valeur

La suppression d'une paire se fait avec \$unset.

```
m$update(query='{"name": "Gandalf"}',  
         update='{"$unset": {"cheveux": "gris"}}',  
         multiple=TRUE) # Tous les documents sont modifiés  
  
m$find(query='{"name": "Gandalf"}', fields='{"_id": 0, "name": 1, "cheveux": 1}')  
  
##      name  
## 1 Gandalf  
## 2 Gandalf  
## 3 Gandalf  
## 4 Gandalf  
## 5 Gandalf
```

Modification - Autres

De nombreuses commandes sont disponibles :

- `$push`, `$pull`, `$addToSet`, ... pour modifier un tableau,
- `$inc`, `$mul`, ... pour modifier une valeur numérique,
- `$rename` pour renommer une clé,
- ...

<https://docs.mongodb.com/manual/reference/operator/update/>

À vous de jouer!

Agrégation et MapReduce

Motivation - Table de contingence

Nous souhaitons compter le nombre de cartes par *Sphere* et par *Type*.

Méthode brutale

```
v_sphere <- m$distinct(key="sphere_name")
v_type <- m$distinct(key="type_name")
tdc <- matrix(0L, nrow=length(v_sphere), ncol=length(v_type),
              dimnames=list(v_sphere, v_type))
for(sphere in v_sphere) {
  for(type in v_type) {
    query <- paste0('{"$and": [{"sphere_name":"' , sphere, '"},',
                      '{"type_name":"' , type, '"}]}')
    tdc[sphere, type] <- m$count(query)
  }
}
print(tdc)
```

Motivation - Table de contingence

Nous souhaitons compter le nombre de cartes par *Sphere* et par *Type*.

Méthode brutale

| # | Hero | Ally | Event | Attachment | Player | Side | Quest | Treasure | Contract |
|---------------|------|------|-------|------------|--------|------|-------|----------|----------|
| ## Leadership | 49 | 68 | 59 | 49 | | 2 | 0 | 0 | |
| ## Tactics | 46 | 62 | 57 | 49 | | 2 | 0 | 0 | |
| ## Spirit | 46 | 64 | 59 | 46 | | 2 | 0 | 0 | |
| ## Lore | 49 | 73 | 57 | 47 | | 2 | 0 | 0 | |
| ## Neutral | 1 | 22 | 22 | 31 | | 2 | 9 | 7 | |
| ## Baggins | 2 | 0 | 2 | 0 | | 0 | 0 | 0 | |
| ## Fellowship | 7 | 0 | 3 | 3 | | 0 | 0 | 0 | |

- C'est très moche!
- Il y a beaucoup d'aller-retours entre le client et le serveur.

Motivation - Table de contingence

Nous souhaitons compter le nombre de cartes par *Sphere* et par *Type*.

Méthode R

```
table(m$find(fields='{"_id": 0, "sphere_name": 1, "type_name": 1}'))
```

| ## ## type_name | sphere_name | | | | | | | |
|----------------------|-------------|---------|------------|------------|------|---------|--------|---------|
| | | Baggins | Fellowship | Leadership | Lore | Neutral | Spirit | Tactics |
| ## Ally | | 0 | 0 | 68 | 73 | 22 | 64 | 62 |
| ## Attachment | | 0 | 3 | 49 | 47 | 31 | 46 | 49 |
| ## Contract | | 0 | 0 | 0 | 0 | 7 | 0 | 0 |
| ## Event | | 2 | 3 | 59 | 57 | 22 | 59 | 57 |
| ## Hero | | 2 | 7 | 49 | 49 | 1 | 46 | 46 |
| ## Player Side Quest | | 0 | 0 | 2 | 2 | 2 | 2 | 2 |
| ## Treasure | | 0 | 0 | 0 | 0 | 9 | 0 | 0 |

Motivation - Table de contingence

Nous souhaitons compter le nombre de cartes par *Sphere* et par *Type*.

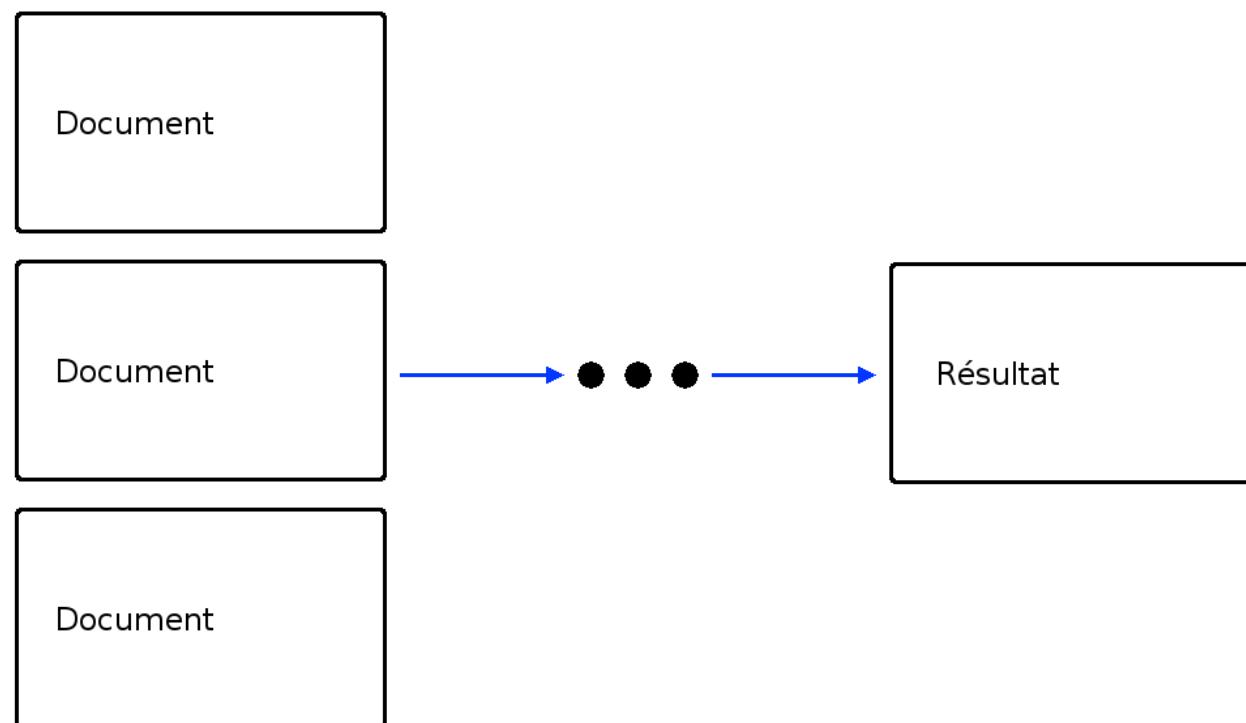
Méthode R

```
table(m$find(fields='{"_id": 0, "sphere_name": 1, "type_name": 1}'))
```

- C'est plus propre!
- Une seule requête mais elle retourne un objet potentiellement volumineux.
- Tous les calculs se font côté client.

Agrégateurs

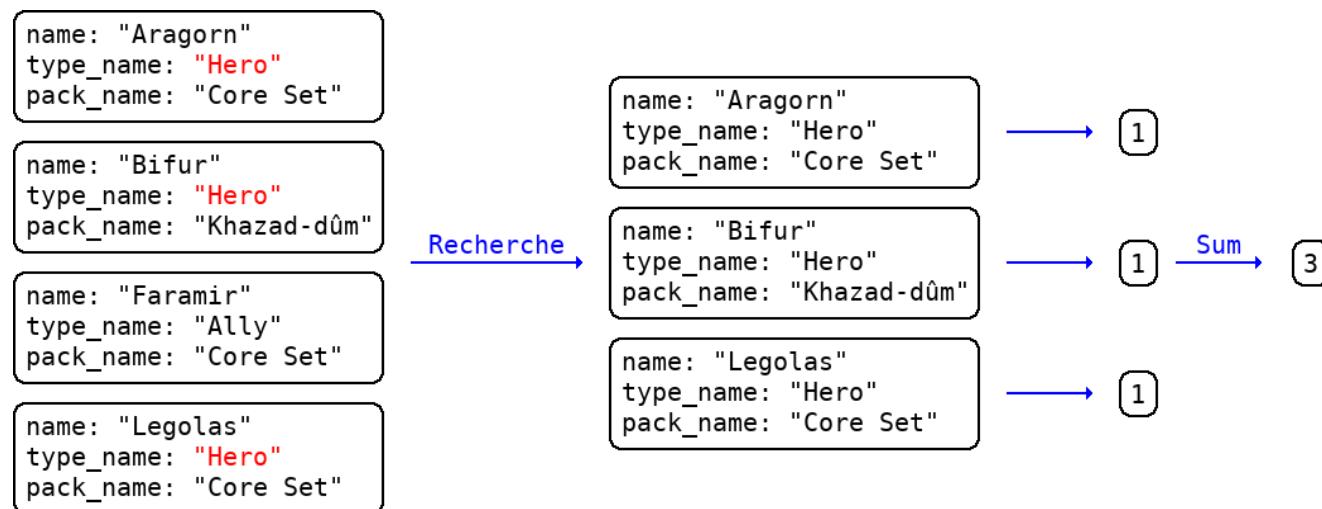
Un *agrégateur* est une fonction qui regroupe les valeurs contenues dans plusieurs documents sélectionnés et retourne une structure contenant des objets “simples” et “plus informatifs”.



Agrégateurs

Les méthodes `count` et `distinct` sont des agrégateurs.

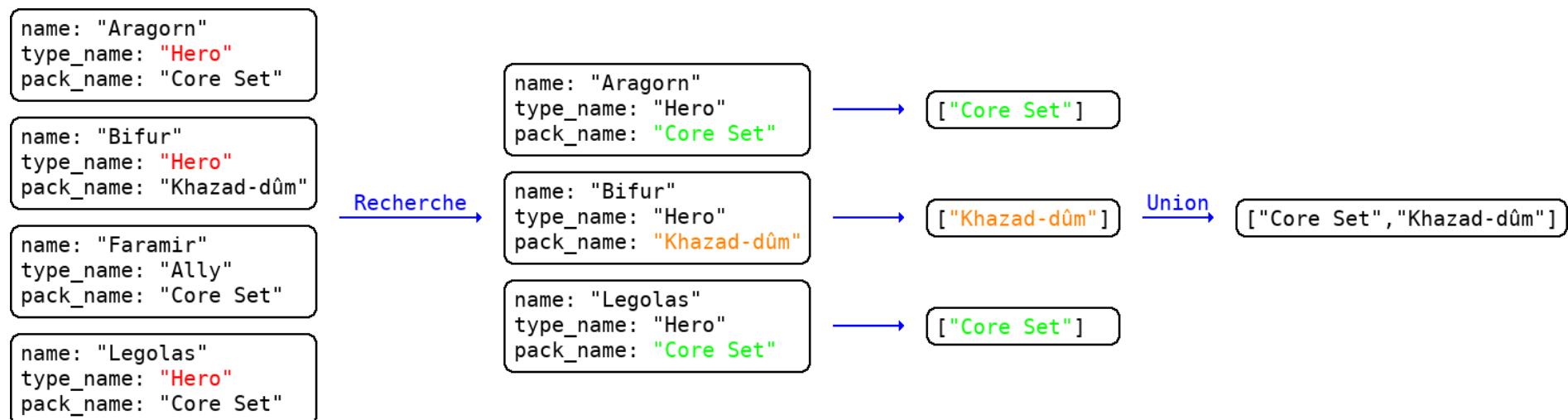
```
m$count('{"type_name": "Hero"}')
```



Agrégateurs

Les méthodes `count` et `distinct` sont des agrégateurs.

```
m$distinct(key="pack_name", query='{"type_name": "Hero"}')
```



Pipeline d'agrégation

Le pipeline d'agrégation de MongoDB est une séquence de **stages** à traverser pour transformer des documents en un résultat agrégé.

Même si nous en verrons un autre dans la suite, il s'agit du modèle d'agrégation privilégié de MongoDB.

Les stages filtrent, transforment, groupent, trient, ... les documents dans un ordre établi. Par exemple :

- `$match` (filtre comme avec `query`),
- `$group` (regroupe et accumule),
- `$sort` (trie).

Pour les autres stages (`$project`, `$limit`, `$skip`, `$sample`, `$out`, ...), voir :

<https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/>

Pipeline d'agrégation - Méthode **aggregate**

La méthode **aggregate** permet de construire un agrégateur basé sur le modèle du pipeline d'agrégation. Cette méthode prend un tableau en paramètre contenant les différentes étapes à réaliser.

Pipeline d'agrégation - Méthode **aggregate**

La méthode **aggregate** permet de construire un agrégateur basé sur le modèle du pipeline d'agrégation. Cette méthode prend un tableau en paramètre contenant les différentes étapes à réaliser.

Illustration avec **count** :

```
m$aggregate('['  
  { "$match": { "type_name": "Hero" } },  
  { "$group": { "_id": null, "count": { "$sum": 1 } } }  
)  
  
## _id count  
## 1 NA 200
```

Pipeline d'agrégation - Méthode **aggregate**

```
m$aggregate('['  
  { "$match": { "type_name": "Hero" } },  
  { "$group": { "_id": null, "count": { "$sum": 1 } } }  
)
```

Quelques explications :

- **\$match** est similaire à **query**,
- le champ **_id** de **\$group** reçoit la clé utilisée pour les groupes ou **null** pour considérer tous les documents,
- le champ **count** de **\$group** est le nom de l'**accumulateur** défini en suivant,
- la valeur des accumulateurs est maintenue groupe par groupe,
- la définition des accumulateurs est évaluée pour chaque document.

Pipeline d'agrégation - Exemple de **distinct**

Pour imiter la méthode **distinct**, nous pouvons utiliser

```
m$aggregate('['  
  { "$group": { "_id": "$sphere_name" } }  
)
```

```
##          _id  
## 1      Baggins  
## 2 Fellowship  
## 3     Neutral  
## 4    Tactics  
## 5 Leadership  
## 6      Spirit  
## 7        Lore
```

Pipeline d'agrégation - Exemple de **distinct**

Pour imiter la méthode **distinct**, nous pouvons utiliser

```
m$aggregate('[  
  { "$group": { "_id": "$sphere_name" } }  
)
```

ou bien

```
m$aggregate('[  
  { "$group": { "_id": null, "sphere_name": { "$addToSet": "$sphere_name" } } }  
)
```

```
##   _id                      sphere_name  
## 1 NA Baggins, Fellowship, Neutral, Tactics, Leadership, Spirit, Lore
```

Pipeline d'agrégation - Tri

Voici comment compter des effectifs et les trier :

```
m$aggregate('[
  { "$group": { "_id": "$sphere_name", "count": { "$sum": 1 } } },
  { "$sort": { "count": -1 } }
]')
```

| | _id | count |
|------|------------|-------|
| ## 1 | Lore | 228 |
| ## 2 | Leadership | 227 |
| ## 3 | Spirit | 217 |
| ## 4 | Tactics | 216 |
| ## 5 | Neutral | 94 |
| ## 6 | Fellowship | 13 |
| ## 7 | Baggins | 4 |

Pipeline d'agrégation - Table de contingence

Pour des effectifs croisés, `_id` est un peu plus compliqué ...

```
m$aggregate('[
  { "$group": { "_id": { "sphere_name": "$sphere_name", "type_name": "$type_name" },
    "count": { "$sum": 1 } } }
]') %>% head(10)
```

| | <code>_id.sphere_name</code> | <code>_id.type_name</code> | count |
|-------|------------------------------|----------------------------|-------|
| ## 1 | Neutral | Contract | 7 |
| ## 2 | Fellowship | Hero | 7 |
| ## 3 | Neutral | Treasure | 9 |
| ## 4 | Baggins | Event | 2 |
| ## 5 | Leadership | Player Side Quest | 2 |
| ## 6 | Tactics | Player Side Quest | 2 |
| ## 7 | Neutral | Hero | 1 |
| ## 8 | Tactics | Attachment | 49 |
| ## 9 | Tactics | Event | 57 |
| ## 10 | Lore | Player Side Quest | 2 |

Pipeline d'agrégation - Opérateurs

Des opérateurs sont disponibles pour calculer des résultats plus avancés.

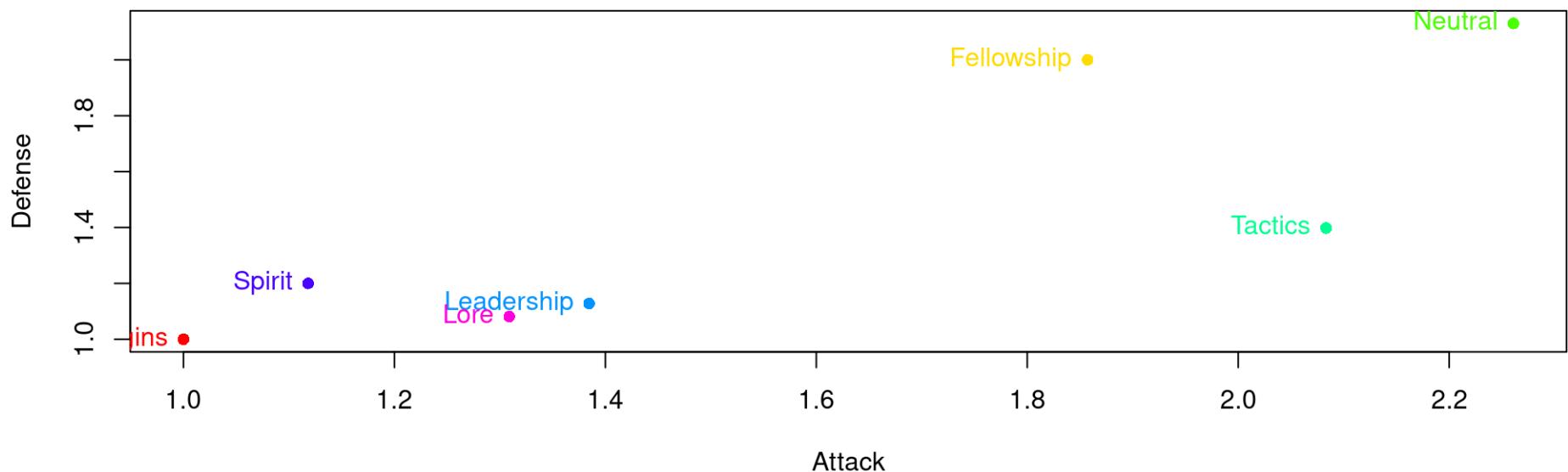
```
m$aggregate(' [  
  { "$group": { "_id": "$sphere_name",  
      "attack": { "$avg": "$attack" },  
      "defense": { "$avg": "$defense" } } }  
)
```

```
##           _id   attack  defense  
## 1     Baggins 1.000000 1.000000  
## 2 Fellowship 1.857143 2.000000  
## 3      Neutral 2.260870 2.130435  
## 4     Tactics 2.083333 1.398148  
## 5 Leadership 1.384615 1.128205  
## 6      Spirit 1.118182 1.200000  
## 7       Lore 1.308943 1.081301
```

Pipeline d'agrégation - Opérateurs

Des opérateurs sont disponibles pour calculer des résultats plus avancés.

```
m$aggregate(' [  
  { "$group": { "_id": "$sphere_name",  
    "attack": { "$avg": "$attack" },  
    "defense": { "$avg": "$defense" } } }  
)
```



Pipeline d'agrégation - Opérateurs

Pour aller plus loin ...

<https://docs.mongodb.com/manual/reference/operator/aggregation/>

```
m$aggregate('[
  { "$match": { "threat": { "$exists": true } } },
  { "$project": { "niveau": { "$cond": [ { "$lt": ["$threat", 10] },
                                            "Faiblard", "Balèze"] } } },
  { "$group": { "_id": "niveau", "count": { "$sum": 1 } } }
]')
```

```
##          _id count
## 1 Faiblard    145
## 2 Balèze      55
```

Pipeline d'agrégation - Limites

Chaque document de la collection résultante est limité à 16MB : il s'agit d'une contrainte liée au format BSON utilisé dans MongoDB.

Chaque stage est limité à 100MB de RAM : cela peut poser des problèmes lorsque les jeux de données sont volumineux.

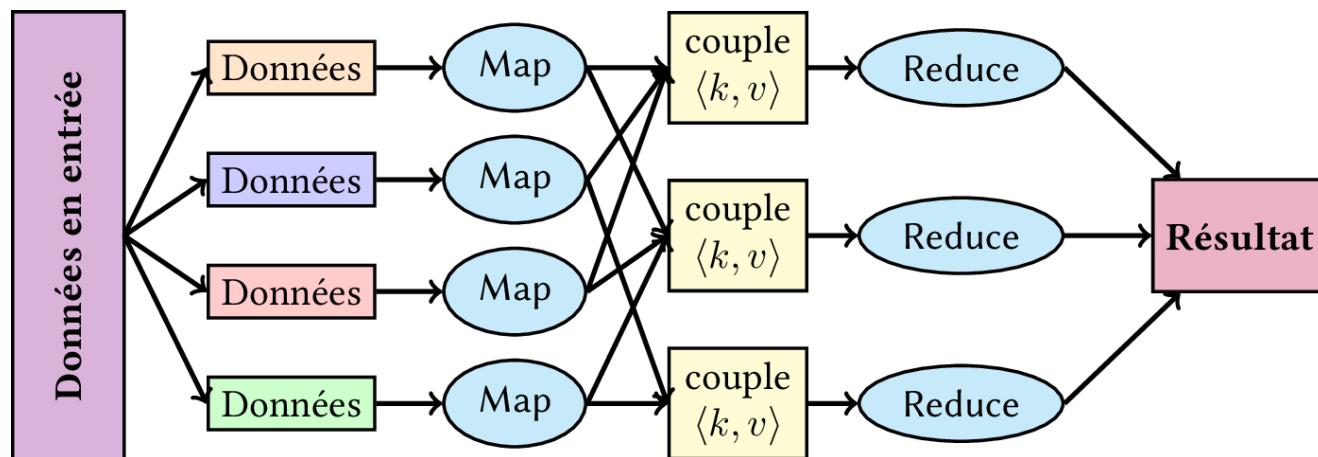
Le pipeline est limité aux opérateurs définis par MongoDB : il n'est pas possible d'utiliser des fonctions plus "souples" pour calculer le résultat d'une agrégation.

MapReduce

MongoDB propose une alternative pour réaliser des agrégations : **MapReduce**

MapReduce est un **patron d'architecture** breveté par Google en 2004. Le principe est de **paralléliser** les calculs qui peuvent alors être **distribués** et de rendre cela invisible à l'utilisateur. Pour cela, MapReduce opère en deux étapes :

- **map** : fonction appliquée à chaque document qui émet une sortie,
- **reduce** : les sorties des appels **map** sont regroupées pour produire le résultat.



MapReduce

Quelques remarques pour MapReduce avec MongoDB :

- la sortie d'un **map** est un document, *i.e.* des paires clé/valeur,
- les fonctions **map** et **reduce** sont écrites en JavaScript (pas en R),
- le nom de la méthode à utiliser est **mapreduce**,
- il est possible de limiter MapReduce au résultat d'une recherche avec le paramètre **query** ou de trier le résultat avec **sort**.

MapReduce offre plus de souplesse que le pipeline d'agrégation mais ce dernier doit rester le choix à privilégier car MapReduce est moins efficace et plus complexe en général.

MapReduce - Compter des effectifs

Considérons un premier exemple simple de MapReduce :

```
m$mapreduce('function() { emit(this.pack_name, 1) }',
             'function(key, values) { return Array.sum(values) }') %>% head(7)
```

| | _id | value |
|------|------------------------|-------|
| ## 1 | A Journey to Rhosgobel | 10 |
| ## 2 | A Shadow in the East | 13 |
| ## 3 | A Storm on Cobas Haven | 11 |
| ## 4 | Across the Ettenmoors | 10 |
| ## 5 | Assault on Osgiliath | 10 |
| ## 6 | Beneath the Sands | 10 |
| ## 7 | Celebrimbor's Secret | 10 |

name: "Aragorn"
type_name: "Hero"
pack_name: "Core Set"

name: "Bifur"
type_name: "Hero"
pack_name: "Khazad-dûm"

name: "Faramir"
type_name: "Ally"
pack_name: "Core Set"

name: "Legolas"
type_name: "Hero"
pack_name: "Core Set"

Map

"Core Set": [1,1,1]
"Khazad-dûm": [1]

Reduce

3
1

MapReduce - Calculer des moyennes

Voici comment obtenir la longueur moyenne des noms en fonction de la *Sphere* :

```
m$mapreduce('function() { emit(this.sphere_name, this.name.length) }',
             'function(key, values) { return Array.sum(values) / values.length }'
)

##          _id    value
## 1      Baggins 14.25000
## 2 Fellowship 12.79592
## 3 Leadership 15.30568
## 4        Lore 16.95340
## 5     Neutral 16.01398
## 6      Spirit 14.54616
## 7   Tactics 14.10471
```

À vous de jouer!