

Introduction au package data.table

B.Thieurmél - benoit.thieurmél@datastorm.fr

Agrocampus Ouest

Introduction

- C'est comme un **data-frame** mais...
- en **plus rapide** dans les requêtes
- et en **plus rapide** dans les calculs !
- avec une **syntaxe particulière**, proche du **SQL** :

```
DT[i, j, by]
```

```
## R:      i      j      by
## SQL: where select | update group by
```

utilise l'objet DT, en sélectionnant les lignes via *i*, en calculant *j*, groupé par *by*

Vignette d'introduction : <https://rawgit.com/Rdatatable/data.table/vignettes/datatable-intro-vignette.html>

Wiki : <https://github.com/Rdatatable/data.table/wiki/Getting-started>

cheatsheet : <https://s3.amazonaws.com/assets.datacamp.com/img/blog/data+table+cheat+sheet.pdf>

Exemples

```
require(data.table)

values = rnorm(500000 * 26)
# un data.frame
df <- data.frame(letters = rep(LETTERS, each = 500000), values = values)

# idem, mais avec data.table
dt <- data.table(letters = rep(LETTERS, each = 1000000), values = values)
# (=) dt <- as.data.table(df)

print(object.size(df), units = 'Mb')
```

```
## 148.8 Mb
```

```
print(object.size(dt), units = 'Mb')
```

```
## 396.7 Mb
```

```
dim(df)
```

```
## [1] 13000000      2
```

- somme des valeurs pour chaque lettre :

```
# lapply sur data.frame
system.time(res1 <- aggregate(values ~ letters, data = df, FUN = sum))
```

```
##    user  system elapsed
##    9.36    0.98   10.36
```

```
# avec data.table
system.time(res2 <- dt[, sum(values), by = letters])
```

```
##    user  system elapsed
##    0.20    0.03    0.23
```

- Ordonner par les valeurs

```
# order sur data.frame
system.time(
  res1 <- df[order(df$values), ]
)
```

```
##    user  system elapsed
##   11.27    0.08   11.37
```

```
# order avec data.table
system.time(
  res2 <- dt[order(values)]
  # = setorder(dt, values)
)
```

```
##    user  system elapsed
##    2.14    0.09    2.23
```

- Faire un subset

```
# data.frame
system.time(
  res1 <- df[df$letters == "M", ]
)
```

```
##    user  system elapsed
##    0.19    0.08    0.27
```

```
# data.table
system.time(
  res2 <- dt[letters == "M", ]
)
```

```
##    user  system elapsed
##    0.06    0.03    0.09
```

Importation

- Utilisation de la fonction `fread` pour l'importation de fichier :

```
# read.table
system.time(
  flights.df <- read.table("flights14.csv", header = T, sep = ",")
)
```

```
##    user  system elapsed
##    1.24    0.03    1.27
```

```
# fread
system.time(
  flights.dt <- fread("flights14.csv")
)
```

```
##      user  system elapsed
##    0.17    0.02    0.19
```

Sélection

- Jeux de données pour les exemples à suivre :

```
set.seed(1234)

dt <- data.table(group = c("A", "B"),
                  cat = rep(c("C", "D"), each = 5000),
                  value = rnorm(10000),
                  weight = 1:10000)

head(dt)
```

```
##      group cat      value weight
## 1:      A   C -1.2070657        1
## 2:      B   C  0.2774292        2
## 3:      A   C  1.0844412        3
## 4:      B   C -2.3456977        4
## 5:      A   C  0.4291247        5
## 6:      B   C  0.5060559        6
```

Sur les lignes

- Pas de rownames dans un **data.table**
- On peut utiliser les indices numériques (*comme avec les data.frame*)
- Ou faire un subset rapide en utilisant les noms de colonnes

```
dt[1:2, ]
dt[c(1,5)] # pas obliger de mettre une virgule...
dt[weight > 9998, ] # pas besoin de "" ou '' pour les noms
dt[order(value)]
```

```
dt[1:2, ]
```

```
##      group cat      value weight
## 1:      A   C -1.2070657        1
## 2:      B   C  0.2774292        2
```

Sur les colonnes

- Sélection numérique **impossible**
- Par défaut, avec les noms de colonnes **sans quote**
- avec **quote** : **data.table** >= 1.10.4 ou utilisation de l'option **with = FALSE**

```
dt[, 1] ## ne marche pas !
dt[, value] ## vecteur
dt[, list(value)] ## data.table
# plusieurs colonnes
dt[, list(group, value)] ## data.table
dt[, .(group, value)] ## raccourci (. == list)
dt[, c("group", "value"), with = FALSE] ## avec des "noms"
## renommage
dt[, list(mygroup = group, myvalue = value)]

##      mygroup      myvalue
## 1:         A -1.2070657
## 2:         B  0.2774292
```

Manipulation

Ajout/Suppression de colonnes

- via l'opérateur := (éviter cbind, peu performant)
- retourne le résultat de façon *invisible*
- suppression avec NULL

```
dt[, tvalue := trunc(value)]
```

```
dt[1:2]
```

```
##      group cat      value weight tvalue
## 1:      A   C -1.2070657      1     -1
## 2:      B   C  0.2774292      2      0
```

- plusieurs colonnes

```
dt[, c("tvalue", "rvalues") := list(trunc(value), round(value, 2))]
```

```
dt[1:2]
```

```
##      group cat      value weight tvalue rvalues
## 1:      A   C -1.2070657      1     -1    -1.21
## 2:      B   C  0.2774292      2      0     0.28
```

- suppression

```
dt[, rvalues := NULL]
```

```
dt[1:2]
```

```
##      group cat      value weight tvalue
## 1:      A   C -1.2070657      1     -1
## 2:      B   C  0.2774292      2      0
```

Modification de colonnes

- via l'opérateur :=, et donc avec un **nom de colonne existante...**

```
dt <- dt[, tvalue := tvalue + 10]

dt[1:2]

##      group cat      value weight tvalue
## 1:      A   C -1.2070657      1      9
## 2:      B   C  0.2774292      2     10

# sur un sous-ensemble de ligne uniquement :
dt <- dt[group == "A", tvalue := tvalue + 100]

dt[1:2]

##      group cat      value weight tvalue
## 1:      A   C -1.2070657      1    109
## 2:      B   C  0.2774292      2     10
```

Calculs

- On peut effectuer tous les calculs directement

```
dt[, sum(value)] # un vecteur

## [1] 61.15893

dt[, list(sum(value))] # un data.table

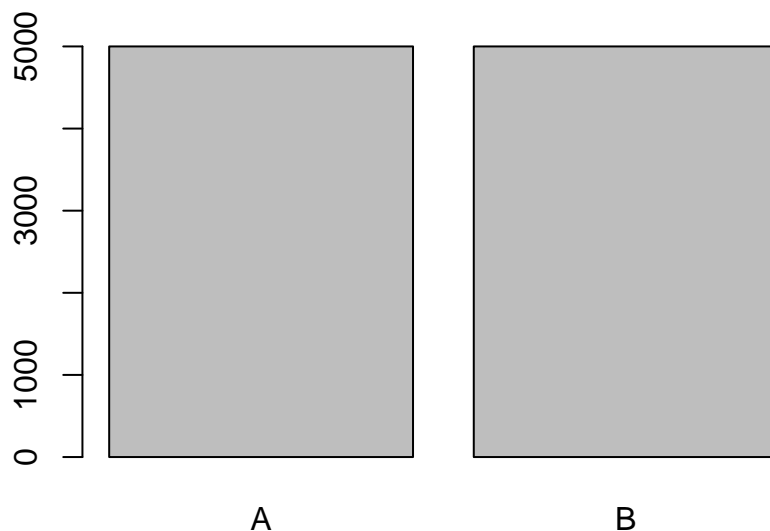
##           V1
## 1: 61.15893

dt[, list(somme = sum(value), moyenne = mean(value))] # calculs multiples + renommage

##      somme      moyenne
## 1: 61.15893 0.006115893
```

- on peut enchaîner plusieurs expressions avec des accolades

```
dt[, {t = table(group)
      barplot(t)
      NULL}]
```



```
## NULL
```

Aggrégation par niveaux

- utilisation du `by`, avec `list()`, ou bien un vecteur de noms

```
# par une variable
```

```
dt[, sum(value), by = group] # dt[, sum(value), by = "group"]
```

```
##      group      V1
```

```
## 1:      A 136.98194
```

```
## 2:      B -75.82302
```

```
# par plusieurs variables et calculs multiples
```

```
dt[, list(somme = sum(value), moy = mean(value)), by = list(group, cat)]
```

```
##      group cat      somme      moy
```

```
## 1:      A   C    2.125355 0.0008501418
```

```
## 2:      B   C   -33.898836 -0.0135595342
```

```
## 3:      A   D   134.856590 0.0539426361
```

```
## 4:      B   D   -41.924180 -0.0167696718
```

```
# dt[, .(somme = sum(value), moy = mean(value)), by = c("group", "cat")]
```

- On peut aussi utiliser des expressions dans le `by`

```
# somme des valeurs, avec un poids inférieur ou supérieur à 5000, par groupe
```

```
dt[, sum(value), by = list(group, weight > 5000)]
```

```
##      group weight      V1
## 1:      A  FALSE    2.125355
## 2:      B  FALSE -33.898836
## 3:      A   TRUE  134.856590
## 4:      B   TRUE -41.924180
```

- by garde l'ordre d'apparition des niveaux
- keyby ordonne le résultat

```
dt[, .(somme = sum(value), moy = mean(value)), keyby = list(group, cat)]
```

```
##      group cat      somme      moy
## 1:      A  C    2.125355 0.0008501418
## 2:      A  D  134.856590 0.0539426361
## 3:      B  C -33.898836 -0.0135595342
## 4:      B  D -41.924180 -0.0167696718
```

-
- .N : retourne le nombre de ligne

```
dt[, .N] # nombre de lignes des données
```

```
## [1] 10000
```

```
dt[, .N, keyby = list(group, cat)] # nombre de lignes pour groupes
```

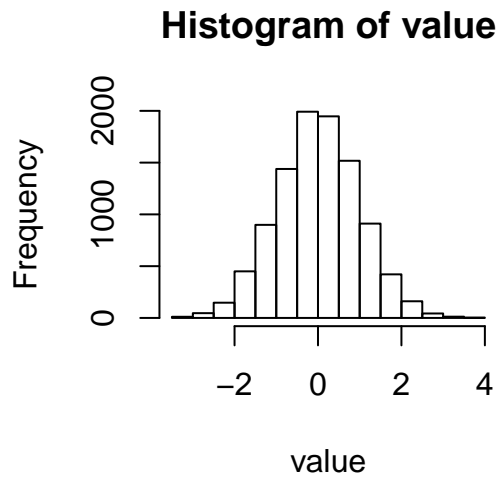
```
##      group cat      N
## 1:      A  C  2500
## 2:      A  D  2500
## 3:      B  C  2500
## 4:      B  D  2500
```

```
dt[group == "A" & cat == "C", list(rows = .N)] # sous sélection
```

```
##      rows
## 1: 2500
```

Graphiques, modèles, ...

```
dt[, {hist(value);NULL}] # suivi d'un NULL, sinon print dans la console...
```



```
## NULL
# graphiques par groupe
par(ask=TRUE)
dt[, hist(value), by = list(group, cat)]
```

Chaînage

- Les opérations sur un **data.table** retourne (en général) un **data.table**
- On peut donc enchaîner les opérations [...]

```
# somme et moyenne des valeurs pour group et cat
# ordonner par somme
dt[, list(somme = sum(value), moy = mean(value)), by = list(group, cat)][order(somme)]
```

```
##      group cat      somme      moy
## 1:      B   D -41.924180 -0.0167696718
## 2:      B   C -33.898836 -0.0135595342
## 3:      A   C   2.125355  0.0008501418
## 4:      A   D 134.856590  0.0539426361
```

.SD

- **.SD** contient toutes les colonnes, à l'exception de celle(s) utilisée(s) dans le by

```
# regardons cela...
dt[1:4, print(.SD), by = .(group)]
```

```
##      cat      value weight tvalue
## 1:      C -1.207066      1     109
## 2:      C  1.084441      3     111
##      cat      value weight tvalue
## 1:      C  0.2774292      2      10
## 2:      C -2.3456977      4       8
```



```
## Empty data.table (0 rows) of 1 col: group
```

-
- on peut l'utiliser pour **faire des calculs sur plusieurs colonnes**

```
dt[, lapply(.SD, mean), by = .(group, cat)]
```

```
##      group cat      value weight  tvalue
## 1:      A   C 0.0008501418   2500 109.9960
## 2:      B   C -0.0135595342   2501   9.9920
## 3:      A   D 0.0539426361   7500 110.0348
## 4:      B   D -0.0167696718   7501   9.9840
```

- une sous-sélection de colonnes est possible avec **.SDcols**

```
dt[, lapply(.SD, mean), by = .(group), .SDcols = "value"] # avec un/des nom(s)
```

```
##      group      value
## 1:      A 0.02739639
## 2:      B -0.01516460
```

```
# ou des indices : dt[, lapply(.SD, mean), by = .(group), .SDcols = 3]
```

les clés

- **data.table** dispose d'un système de clés
- le tableau est alors ordonné par les clés
- les subsets sur les clés seront plus performants
- argument *key* dans la fonction **data.table**
- ou **setkey** avec des noms de colonnes sans quote
- **setkeyv** avec avec quote
- **key** pour connaître les clés de la table

```
set.seed(1234)
values = rnorm(384616 * 26)
dt <- data.table(letters = rep(LETTERS, each = 384616), group = letters[1:16], values = values)
```

-
- la sélection est plus rapide, et l'appel simplifié, **par défaut dans l'ordre des clés**

```
setkey(dt, NULL)
system.time(
  dt[group == "f", ]
)
```

```
##      user  system elapsed
##      0.06    0.02    0.09
```

```
setkey(dt, group)
system.time(
  dt["f", ]
)
```

```
##      user  system elapsed
##      0.04    0.00    0.03
```

- clés multiples : sélection via une *liste*

```
setkey(dt, NULL)
system.time(
  dt[letters == "M" & group == "f", ]
)
```

```
##      user  system elapsed
##    0.08    0.01    0.09
```

```
setkey(dt, letters, group)
system.time(
  dt[list("M", "f"), ]
)
```

```
##      user  system elapsed
##         0         0         0
```

-
- clés et valeurs multiples : sélection via une *liste*, et des *vecteurs* de valeurs

```
setkey(dt, NULL)
system.time(
  dt[letters == "M" & group %in% c("f", "g"), ]
)
```

```
##      user  system elapsed
##    0.19    0.06    0.25
```

```
setkey(dt, letters, group)
system.time(
  dt[list("M", c("f", "g")), ]
)
```

```
##      user  system elapsed
##         0         0         0
```

Transformation

- Via deux fonctions `melt`, et `dcast`, basées sur celles présentes dans le package **reshape2**

```
# les données
dt <- data.table(airquality)
dt
```

```
##      Ozone Solar.R Wind Temp Month Day
##    1:    41     190  7.4   67     5   1
##    2:    36     118  8.0   72     5   2
##    3:    12     149 12.6   74     5   3
##    4:    18     313 11.5   62     5   4
##    5:    NA      NA 14.3   56     5   5
## ---
## 149:    30     193  6.9   70     9  26
## 150:    NA     145 13.2   77     9  27
## 151:    14     191 14.3   75     9  28
## 152:    18     131  8.0   76     9  29
## 153:    20     223 11.5   68     9  30
```

melt

```
res_melt <- melt(data = dt, id = c("Month", "Day"))
res_melt
```

```
##      Month Day variable value
##    1:     5   1    Ozone    41
##    2:     5   2    Ozone    36
##    3:     5   3    Ozone    12
##    4:     5   4    Ozone    18
##    5:     5   5    Ozone    NA
## ---
## 608:     9  26     Temp     70
## 609:     9  27     Temp     77
## 610:     9  28     Temp     75
## 611:     9  29     Temp     76
## 612:     9  30     Temp     68
```

dcast

```
res_dcast <- dcast.data.table(data = res_melt, Month + Day ~ variable)
res_dcast
```

```
##      Month Day Ozone Solar.R Wind Temp
##    1:     5   1    41     190  7.4   67
##    2:     5   2    36     118  8.0   72
##    3:     5   3    12     149 12.6   74
##    4:     5   4    18     313 11.5   62
##    5:     5   5     NA       NA 14.3   56
## ---
## 149:     9  26     30     193  6.9   70
## 150:     9  27     NA     145 13.2   77
## 151:     9  28     14     191 14.3   75
## 152:     9  29     18     131  8.0   76
## 153:     9  30     20     223 11.5   68
```

Pour aller plus loin...

- Vignette d'introduction : <https://rawgit.com/wiki/Rdatatable/data.table/vignettes/datatable-intro-vignette.html>
- Semantique : <https://rawgit.com/wiki/Rdatatable/data.table/vignettes/datatable-reference-semantics.html>
- Clés : <https://rawgit.com/wiki/Rdatatable/data.table/vignettes/datatable-keys-fast-subset.html>
- Transformations : <https://rawgit.com/wiki/Rdatatable/data.table/vignettes/datatable-reshape.html>

Mais surtout pratiquer !