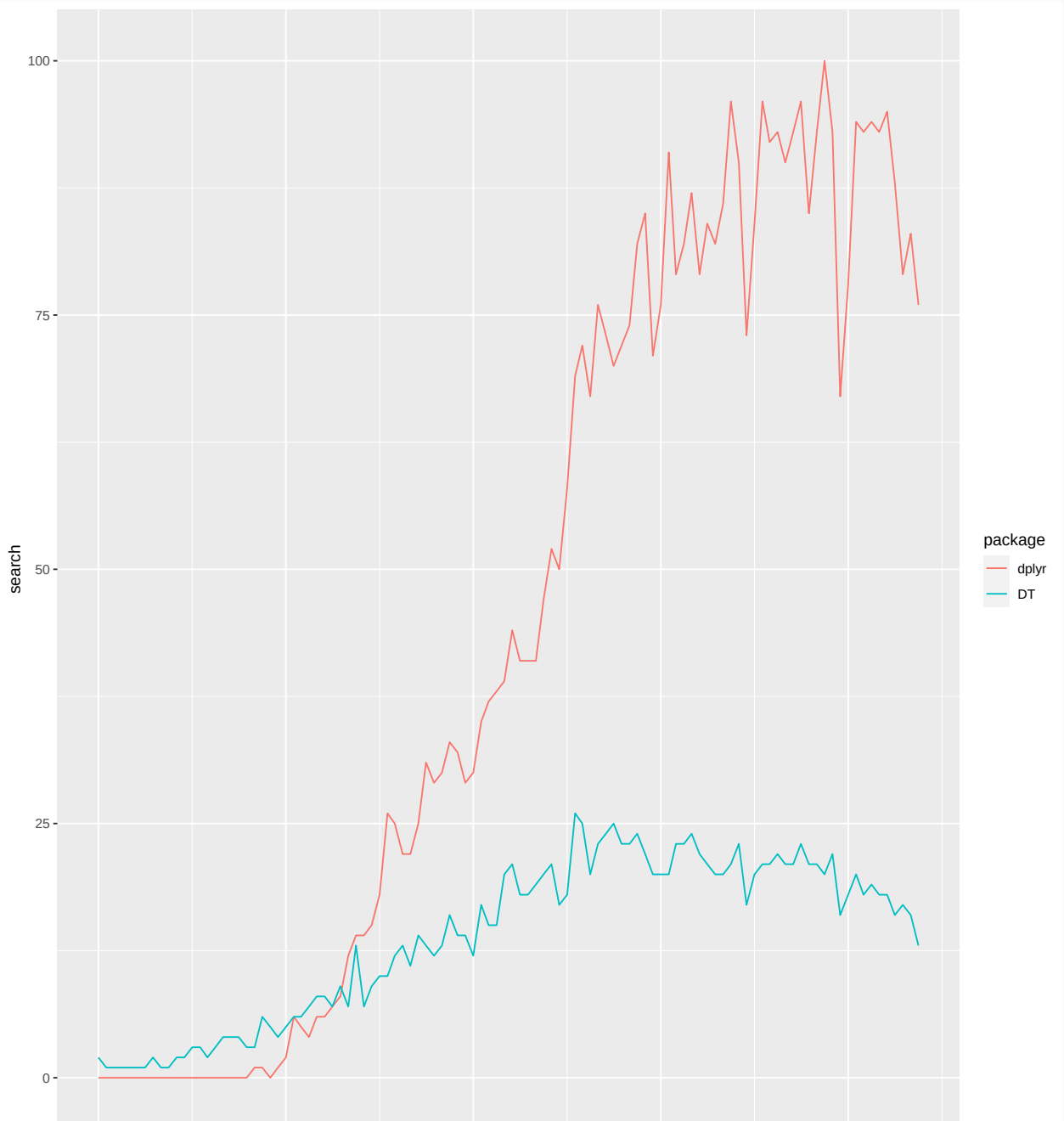
Intro

`data.table`, mais à quoi ça sert

1. A rien ;
2. A frimer ;
3. A refaire du `dplyr` différemment ;
4. A apprendre des choses nouvelles ;
5. A manipuler des données (en mémoire) rapidement.

Comparaison graphique



Data Table

- C'est comme un data-frame
- Plus rapide dans les requêtes **et** plus rapide dans les calculs
- Syntaxe un peu particulière

```
DT[i, j, by]

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

utilise DT, sélectionne

- les lignes via i
- calcule j
- en groupant par by

Premiers pas

Création à la main

```
install.packages("data.table") ; library(data.table)
set.seed(1234)
rx <- rnorm(6)
print(rx)
[1] -1.2070657  0.2774292  1.0844412 -2.3456977
[5]  0.4291247  0.5060559

f <- gl(3, 2, labels=c("g1","g2","g3"))
print(f)
[1] g1 g1 g2 g2 g3 g3
Levels: g1 g2 g3

ch <- c(rep("A",3),rep("B",3))
ch
[1] "A" "A" "A" "B" "B" "B"
```

data-frame df

```
df <- data.frame(x=rx, f, ch)
df
           x  f ch
1 -1.2070657 g1  A
2  0.2774292 g1  A
3  1.0844412 g2  A
4 -2.3456977 g2  B
5  0.4291247 g3  B
6  0.5060559 g3  B

       x             f          ch
 Min.   :-2.3457   g1:2   Length:6
 1st Qu.:-0.8359   g2:2   Class :character
 Median : 0.3533   g3:2   Mode  :character
 Mean   :-0.2093
 3rd Qu.: 0.4868
 Max.   : 1.0844
```

data-table dt

```
dt <- data.table(x=rx, f, ch)
dt
             x  f ch
1: -1.2070657 g1  A
2:  0.2774292 g1  A
3:  1.0844412 g2  A
4: -2.3456977 g2  B
5:  0.4291247 g3  B
6:  0.5060559 g3  B

       x             f          ch
 Min.   :-2.3457   g1:2   Length:6
 1st Qu.:-0.8359   g2:2   Class :character
 Median : 0.3533   g3:2   Mode  :character
 Mean   :-0.2093
 3rd Qu.: 0.4868
 Max.   : 1.0844
```

conversion en data-table / suppression

Un peu comme d'habitude, la conversion

```
dt1 <- as.data.table(df) #(voir aussi setDT)
```

Comme d'habitude

```
rm(dt1)
```

Sélection de colonnes i

1. avec les Noms

```
dt[,c("x","f")]
             x  f
1: -1.2070657 g1
2:  0.2774292 g1
3:  1.0844412 g2
4: -2.3456977 g2
5:  0.4291247 g3
6:  0.5060559 g3
```

Sélection de colonnes ii

2. avec les Indices

```
dt[,c(2,1,1)]
    f           x           x
1: g1 -1.2070657 -1.2070657
2: g1  0.2774292  0.2774292
3: g2  1.0844412  1.0844412
4: g2 -2.3456977 -2.3456977
5: g3  0.4291247  0.4291247
6: g3  0.5060559  0.5060559
```

3. avec les Logiques : NON

```
dt[,c(F, T, T)]
[1] FALSE  TRUE  TRUE
```

Sélection de lignes i

1. Noms

```
rownames(dt)
[1] "1" "2" "3" "4" "5" "6"

dt[c("1","2"),] # Error
```

2. Lignes

```
dt[c(1,3,1),]
#+RESULTS:
             x  f ch
1: -1.207066 g1  A
2:  1.084441 g2  A
3: -1.207066 g1  A
```

Sélection de lignes ii

3. Logiques

```
dt[c(F,F,F,T,T,F),]
             x  f ch
1: -2.3456977 g2  B
2:  0.4291247 g3  B
```

Conclusion
Tout est (presque) pareil

Détails

- rownames et affichage

```
rownames(dt) <- paste("ind",1:6,sep="")
dt
             x  f ch
1: -1.2070657 g1  A
2:  0.2774292 g1  A
...

rownames(dt)
[1] "ind1" "ind2" "ind3" "ind4" "ind5" "ind6"
```

Détails (suite)

- Sélection d'une colonne

```
df[,1]
[1] -1.2070657  0.2774292  1.0844412...

dt[,1]
1: -1.2070657
2:  0.2774292
...
```

Fusion de tableaux par clef

```
dt2 <- data.table(ch2=c("A", "B"), y=c(0,1))
dt2
       ch2 y
    1:   A 0
    2:   B 1

merge(dt,dt2,by.x="ch",by.y="ch2")
   ch           x  f y
1:  A -1.2070657 g1 0
2:  A  0.2774292 g1 0
3:  A  1.0844412 g2 0
4:  B -2.3456977 g2 1
5:  B  0.4291247 g3 1
6:  B  0.5060559 g3 1
```

Concatenation de tableaux (colonne)

```
dt1 <- data.table(ch2=rep(c("A", "B"),3),
         y=rep(c(0,1), each=3))
cbind(dt,dt1)
```

```
            x  f ch ch2 y
1: -1.2070657 g1  A   A 0
2:  0.2774292 g1  A   B 0
3:  1.0844412 g2  A   A 0
4: -2.3456977 g2  B   B 1
5:  0.4291247 g3  B   A 1
6:  0.5060559 g3  B   B 1
```

Concatenation de tableaux (ligne)

```
dt3 <- data.table(x=0, f="g4", ch="C")
res <- rbind(dt,dt3)
summary(res)
```

```
       x                f          ch
 Min.   :-2.3457   g1:2   Length:7
 1st Qu.:-0.6035   g2:2   Class :character
 Median : 0.2774   g3:2   Mode  :character
 Mean   :-0.1794   g4:1
 3rd Qu.: 0.4676
 Max.   : 1.0844
```

Unique/duplicated/tri

```
res <- rbind(dt,dt)
unique(res)
duplicated(res)
```

pour faire la transition vers les commandes spécifiques

```
uniqueN(res)
[1] 6
```

ordonner (voir aussi setorder)

```
dt[order(f,ch),]
dt[order(-f,-ch),]
```

Spécificités

Sélection de colonnes

selection par liste

On prend une liste d'objets : les "colonnes"

ce n'est ni un vecteur de caractères des noms de colonnes, ni une liste de caractères

```
dt[,list(x,f)]
dt[,.(x,f)]
             x  f
1: -1.2070657 g1
2:  0.2774292 g1
3:  1.0844412 g2
4: -2.3456977 g2
5:  0.4291247 g3
6:  0.5060559 g3
```

Sélection d'une colonne

```
dt[,.(x)]
             x
   1: -1.2070657
   2:  0.2774292
   3:  1.0844412
   4: -2.3456977
   5:  0.4291247
   6:  0.5060559
dt[,x]
[1] -1.2070657  0.2774292  1.0844412 -2.3456977
[5]  0.4291247  0.5060559
```

Sélection de lignes

```
dt[x>0,]
              x  f ch
   1: 0.2774292 g1  A
   2: 1.0844412 g2  A
   3: 0.4291247 g3  B
   4: 0.5060559 g3  B
dt[(x>0) & (ch=="A"),]
              x  f ch
   1: 0.2774292 g1  A
   2: 1.0844412 g2  A
```

Opérateurs

`& | == != < <= > >= is.na() %in% !`

`%like% %between%`

Création de colonne

```
dt[,"n":=1:6]
dt
```

```
                x  f ch n
   1: -1.2070657 g1  A 1
   2:  0.2774292 g1  A 2
   3:  1.0844412 g2  A 3
   4: -2.3456977 g2  B 4
   5:  0.4291247 g3  B 5
   6:  0.5060559 g3  B 6
```

Suppression de colonne

```
dt[,"n":=NULL]
dt
                x  f ch
   1: -1.2070657 g1  A
   2:  0.2774292 g1  A
   3:  1.0844412 g2  A
   4: -2.3456977 g2  B
   5:  0.4291247 g3  B
   6:  0.5060559 g3  B
```

ou pour le coté pratique, pour une colonne

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

Création de colonneS

```
dt[,c("n","m"):=list(1:6,2:7)]
dt
                x  f ch n
   1: -1.2070657 g1  A 1
   2:  0.2774292 g1  A 2
   3:  1.0844412 g2  A 3
   4: -2.3456977 g2  B 4
   5:  0.4291247 g3  B 5
   6:  0.5060559 g3  B 6
```

Méthode “fonctionnelle”

```
dt[,':='(n=1:6, m=2:7)]
```

Suppression de colonneS

```
dt[,c("n","m"):=list(NULL,NULL)]
dt
                x  f ch
   1: -1.2070657 g1  A
   2:  0.2774292 g1  A
```

ou

```
dt[,c("n","m"):=NULL] # recyclage
```

ou encore

```
nom <- c("n","m")
dt[,c(nom):=NULL]
```

Attention ci-dessous cela cherche à éliminer la colonne nom

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

Remplacement de valeurs (modification)

```
dt[ch=="A",f:="g1"]
dt
                x  f ch
   1: -1.2070657 g1  A
   2:  0.2774292 g1  A
   3:  1.0844412 g1  A
   4: -2.3456977 g2  B
   5:  0.4291247 g3  B
   6:  0.5060559 g3  B
```

Remplacement de valeurs (modification)

Tri en place (le tableau est modifié)

```
setorder(dt,f,-ch)
dt
             x  f ch
1: -1.2070657 g1  A
2:  0.2774292 g1  A
3: -2.3456977 g2  B
4:  1.0844412 g2  A
5:  0.4291247 g3  B
6:  0.5060559 g3  B
```

Faire des calculs sur les variables

- Faire un comptage

```
dt[,.N] ## nombre de lignes
dt[f=="g1",.N] ## nbe lignes dont f est g1
```

- Faire une numérotation

```
dt[,.I] ## numérote les observations
```

Calculs simples

```
mode <- function(x) {
  t <- table(x)
  return(names(t)[which.max(t)]) }

dt[,.(som=sum(x),med=median(x),mod=mode(f))]
           som       med mod
1: -1.255712 0.353277  g1
```

C'est un calcul, il n'est pas ajouté à dt

Calculs plus complexes

```
set.seed(2145)
dt[, "y":=2*x +1 + rnorm(nrow(dt))]

dt[,as.list(coef(lm(y~x)))]
   (Intercept)        x
1:    1.230908 2.005857
```

Attention
Le résultat est une liste (pas un vecteur)

Autres exemples

```
dt[,plot(y~x)]

dt[,c(as.list(coef(lm(y~x))), lm(y~x)$df.residual)]
   (Intercept)        x V3
1:    1.230908 2.005857  4
```

Exemple de l'aide

```
> DT=data.table(ID=rep(c("b","a","c"),times=3:1),
+   a=1:6,b=7:12,c=13:18)
   ID a  b  c
1:  b 1  7 13
2:  b 2  8 14
3:  b 3  9 15
4:  a 4 10 16
5:  a 5 11 17
6:  c 6 12 18
```

Opérations sur toutes les colonnes

```
DT[, lapply(.SD, mean)],
   ID   a   b    c
1: NA 3.5 9.5 15.5
Warning message:
In mean.default(X[[i]], ...) :
  argument is not numeric or logical: returning NA

DT[, lapply(.SD, mean)], .SDcols = names(DT)[-1]]
   ID   a    b    c
1:  b 2.0  8.0 14.0
2:  a 4.5 10.5 16.5
3:  c 6.0 12.0 18.0

DT[, lapply(.SD, quantile), .SDcols = names(DT)[-1]]
```

deep et shallow copy

- Copions le data-frame dans un nouvel objet

```
df1 <- df
lobstr::obj_addr(df)
[1] "0x55c8f674e788"
lobstr::obj_addr(df1)
[1] "0x55c8f674e788"
```

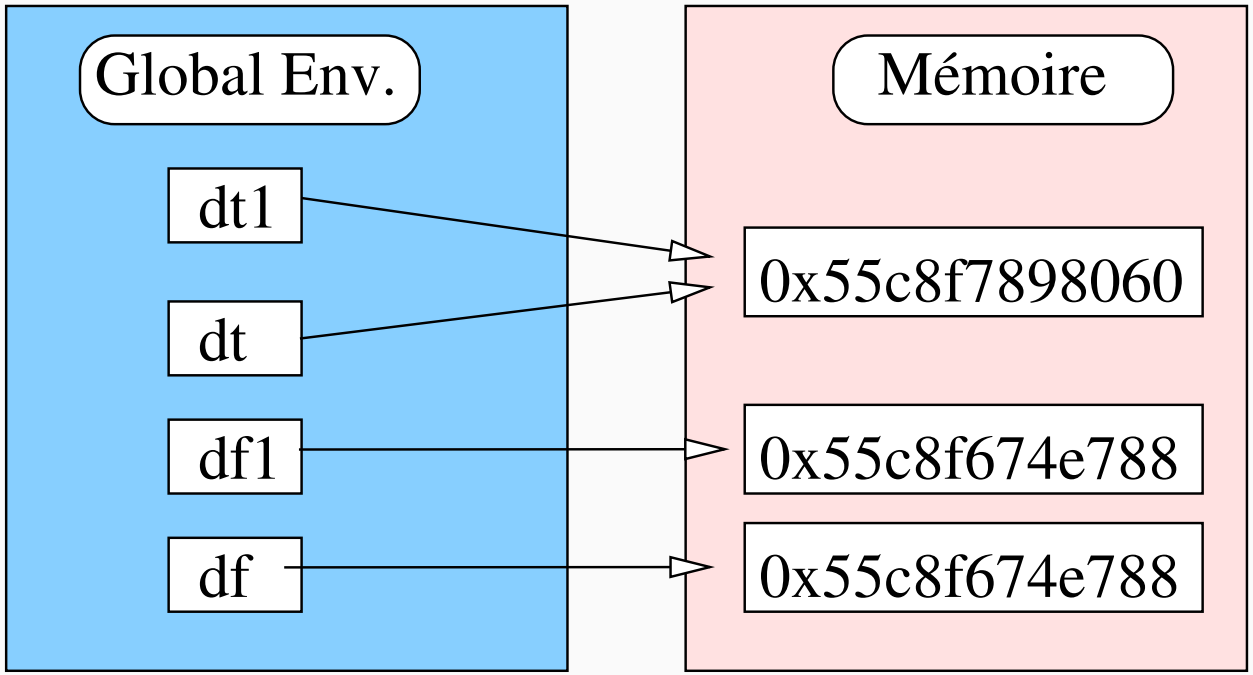
Les objets pointent sur 2 adresses $\neq$ (appelée deep copy)

- Copions le data-table dans un nouvel objet

```
dt1 <- dt
lobstr::obj_addr(dt)
[1] "0x55c8f7898060"
lobstr::obj_addr(dt1)
[1] "0x55c8f7898060"
```

Les objets pointent sur la même adresse (shallow copy)

deep et shallow copy



exemple deep et shallow copy

Implications pratiques

```
names(dt)
[1] "x"  "f"  "ch"

dt[,"z":=1:6]
dt[,"z":=1:6]
names(dt1)
[1] "x"  "f"  "ch" "z"
```

Donc quand on veut faire une (deep) copie, on utilise

```
dt2 <- copy(dt)
```

syntaxe

in place

- :=
- fonction commençant par set

```
dt_new <- as.data.table(df) ## doublon
dt <- setDT(df) ## a utiliser

df_new <- as.data.frame(dt) ## doublon
df <- setDF(dt) ## a utiliser

setnames(dt, c("f","ch"), c("varqual", "caract"))
```

Agrégations : par niveau de ...

```
dt[,.N,by=.(f)] ## combien de lignes par modalite de f
dt[,.N,by="f"]  ## c
dt[,.GRP, by=.(f)] ##numérote les groupes
##   (1= g1, 2=g2, 3=g3)

## age moyen par ville quartier
dtf[,agem=mean(age),by=c("ville","quartier")]
dtf[,agem=mean(age),by=.(ville,quartier)] # idem

## résultat ordonné
dtf[,agem=mean(age),keyby=.(ville,quartier)]
```

On peut aussi utiliser des booleens pour faire le by

```
ans <- flights[, .N, .(dep_delay>0, arr_delay>0)]
ans
    dep_delay arr_delay      N
 1:      TRUE      TRUE  72836
 2:     FALSE      TRUE  34583
 3:     FALSE     FALSE 119304
 4:      TRUE     FALSE  26593
```

Opérations sur toutes les colonnes (le retour)

```
> DT[, lapply(.SD, mean), by=.(ID)]
   ID   a    b    c
1:  b 2.0  8.0 14.0
2:  a 4.5 10.5 16.5
3:  c 6.0 12.0 18.0

> DT[, lapply(.SD, quantile), by=ID]
```

objet.SD

Crée une liste SANS les colonnes du by

```
> DT[, print(.SD), by=ID]
   a b  c
1: 1 7 13
2: 2 8 14
3: 3 9 15
   a  b  c
1: 4 10 16
2: 5 11 17
   a  b  c
1: 6 12 18
Empty data.table (0 rows) of 1 col: ID
```

objet .SD

Composantes
de la liste
[[1]] [[2]] [[3]]

	a	b	c
1:	1	7	13
2:	2	8	14
3:	3	9	15

Groupe ID 1

	a	b	c
1:	4	10	16
2:	5	11	17

Groupe ID 2

	a	b	c
1:	6	12	18

Groupe ID 3

Opération via .SD en sélectionnant les colonnes

```
> DT[, lapply(.SD, mean), by=ID,.SDcols = c("a","b")]
   ID   a    b
1:  b 2.0  8.0
2:  a 4.5 10.5
3:  c 6.0 12.0
```

Chainage

On peut enchainer des opérations de []

```
dt[,.N,by=.(f)]
     f N
 1: g1 2
 2: g2 2
 3: g3 2

dt[,.(eff=.N),by=.(f)][,mean(eff)]
[1] 2 ## <- pourquoi cette présentation ?
```

Tables dans la session

```
tables()
     NAME NROW NCOL MB   COLS KEY
1:    dt    6    3  0 x,f,ch
Total: 0MB
```

Index sur une colonne

Pour effectuer une sélection des lignes et calculs par groupe rapide : créer un index.

```
setkey(dt,f) ## création d'un index
tables()
     NAME NROW NCOL MB   COLS KEY
  1:   dt    6    3  0 x,f,ch   f
```

1. Tri selon la clef
2. Marque les colonnes comme “key columns” (attribut sorted au dt)

Sélection de ligne par index

```
dt["g1",] ## ou mult="all"
                x  f ch
  1: -1.2070657 g1  A
  2:  0.2774292 g1  A
dt[.("g1"),mult="all"]
```

Sélection de ligne et option

```
dt["g1",mult="first"]
            x  f ch
  1: -1.207066 g1  A
dt["g1",mult="last"]
            x  f ch
  1: 0.2774292 g1  A
dt["g1",sum(x)]
[1] -0.9296365
```

```
dt[.(c("g1","g2")),] ## ou dt[c("g1","g2"),]
             x  f ch
1: -1.2070657 g1  A
2:  0.2774292 g1  A
3:  1.0844412 g2  A
4: -2.3456977 g2  B

dt[f%in%c("g1","g2"),] ## idem
```

Index composite

Sur plusieurs colonnes

```
setkey(dt, ch, f)
tables()
   NAME NROW NCOL MB   COLS  KEY
1:   dt    6    3  0 x,f,ch ch,f

dt[list("A","g1"),]
             x  f ch
1: -1.2070657 g1  A
2:  0.2774292 g1  A
```

1. D'abord data-table cherche les valeurs de la variable `ch` qui correspondent à `"A"`.
2. Puis, parmi ces lignes, il recherche celles dont la variable `f` vaut g1

Index multiple et sélection

Sur une colonne parmi celles constituant l'index :

```
dt[list("A"),]
             x  f ch
1: -1.2070657 g1  A
2:  0.2774292 g1  A
3:  1.0844412 g2  A

dt[list(unique(ch),"g1"),]
             x  f ch
1: -1.2070657 g1  A
2:  0.2774292 g1  A
3:         NA g1  B
```

La seconde forme car il faut d'abord sélectionner par la première variable de l'index (sélection donc sur toutes les valeurs possibles)

Jointures

Pour les jointures voir

- `dt1[dt2, on=.(f,ch)]`
- Rolling joins
- Overlap joins

I/O

- les fonctions `fread` et `fwrite` permettent de faire des entrees/sorties sur disque au format texte avec un support du format CSVY `https://csvy.org/`
- entrée vers 1 data-table à partir d'une liste R `rbindlist`

Un exemple

Les données

```
r,"V1","V2","V3","V4","V5","V6","V7","V8","V9",
"V10","V11","V12","V13","V14","V15","V16","V17",
"V18","V19","V20","V21","V22","V23","V24","V25",
"V26","V27","V28","V29"
"1","age",5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
20,21,22,23,24,30,36,42,48,54,62,68,74
"2","hauteur.indiv1",2.7186407226455,3.33601595749094,
3.57902542042516,4.37971696820213,5.6707707427398 3,...
"3","hauteur.indiv2",0.554447008369346,1.766910316 1036,
3.24258705362362,6.896779355 5934,6.9488342049 8552,
4.97362286486496,6.54974596604283,7.0661733958789,
8.0572302703491,9.01087793104943,...
```

import

L'import via `fread` nous donne

```
dt <- fread("hauteur.csv", drop=1, header=TRUE, skip=1)
dt
                age         5        6         7        8
1: hauteur.indiv1 2.7186407 3.336016 3.5790254 4.379717...
2: hauteur.indiv2 0.5544470 1.766910 3.2425870 6.896779...
```

Objectif

On veut

```
individu,age,ht
1,5,2.7186407226455
1,6,3.33601595749094
1,7,3.57902542042516
...
2,5,0.554447008369346
2,6,1.7669103161036
2,7,3.24258705362362
```

Préliminaire changer les noms des variables

```
setnames(dt, names(dt)[1], "individu")
          individu         5        6         7        8
1: hauteur.indiv1 2.7186407 3.336016 3.5790254 4.379717
2: hauteur.indiv2 0.5544470 1.766910 3.2425871 6.896779
```

Puis on change le contenu de la variable individu

```
dt[,individu:=substr(individu,14,14)]
dt
  individu         5        6         7        8
1:        1 2.7186407 3.336016 3.5790254 4.379717
2:        2 0.5544470 1.766910 3.2425871 6.896779
```

Large vers long : finalisation

Le passage vers Long avec `melt`

```
dtl <- melt(dt, id.vars="individu", measure.vars=
 as.character(c(5,6,7,8,9,10,11,12,13,14,15,16,17,18,
  19,20,21,22,23,24,30,36,42,48,54,62,68,74)))
dtl
     individu variable      value
  1:        1        5  2.7186407
  2:        2        5  0.5544470
  3:        3        5  2.2337488
## changement de nom
setnames(dt, names(dtl)[2:3], c("age","ht"))
     individu age         ht
  1:        1   5  2.7186407
  2:        2   5  0.5544470
  3:        3   5  2.2337488
```

Long vers large : départ

Un tableau

```
rats <- fread("rats.csv")
rats[1:8,]
    sujet   dose  taille age
 1:     8 faible 71.7008  50
 2:     8 faible 78.8162  60
 3:     8 faible      NA  70
 4:     8 faible      NA  80
 5:     8 faible      NA  90
 6:     8 faible      NA 100
 7:     8 faible      NA 110
 8:    10 faible 71.2811  50
...
```

Long vers large : objectif

Nous voulons

```
indiv   dose taille50 taille60 ... taille110
    8 faible  71.7008  78.8162  ...       NA
   10 faible  71.2811 ...
```

Long vers large : dcast

La syntaxe

```
dcast(rats, sujet~age, value.var="taille")
    sujet       50       60       70       80       90...
 1:     1 69.3397 73.2462 77.3886 78.1025 76.4853...
 2:     3 75.0067 78.5684 79.9250 80.7775       NA...
```

La formule `LHS ~ RHS` permet de spécifier

- à gauche (LHS) ce qui spécifie la ligne : le numéro du sujet
- à droite ce qui spécifie les colonnes : les différents ages

et `value.var="taille"` spécifie dans quelle variable on trouve les valeurs à mettre.