

1. **Overview:**

* It belongs to family of data frames.
* It avoids allocating memory to the intermediate steps such as filtering.
* It creates **shallow** copies (i.e. data is not physically copied in systems memory). It's just a copy of column pointers.
* It enhanced the speed of indexing, rolling ordered joins, overlapping range joins, assignment, grouping and listing columns.
* It uses **radix** **sort** to do the sorting.
* Almost all the operations are **20x faster** than **dplyr** (the fastest library till date).
* Syntax like SQL

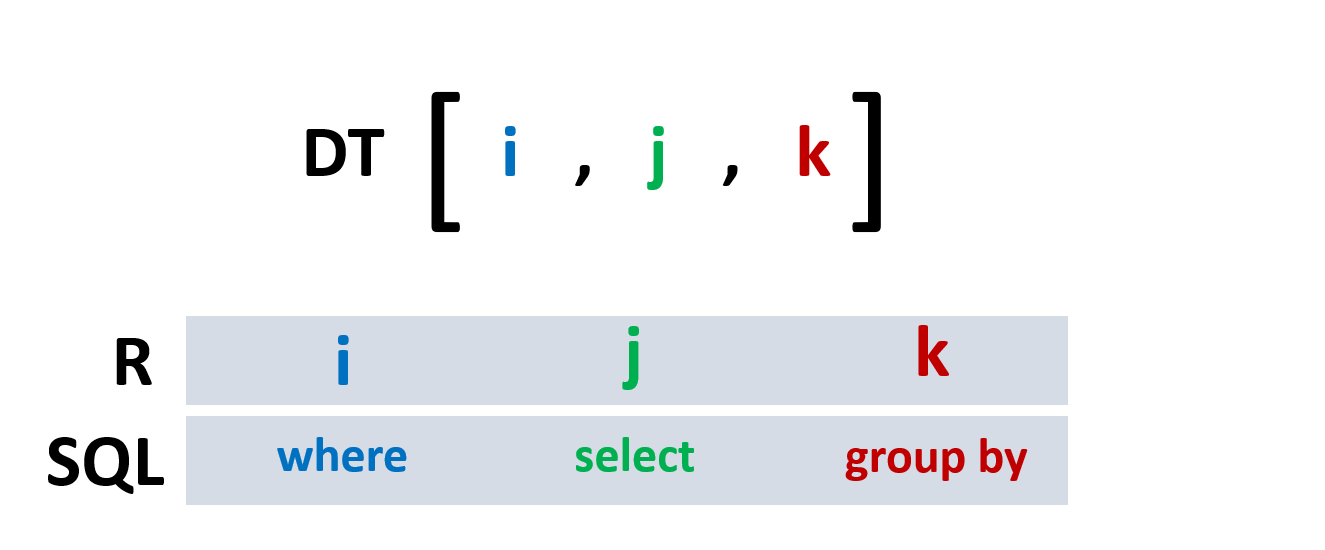
1. **Beginner Course:**

https://www.datacamp.com/courses/data-table-data-manipulation-r-tutorial

1. **Installation:**

install.packages('data.table')

1. **Syntax:**



where,

* DT = a sample data.table
* i = similar to **'where'** in SQL
* j = similar to **'select'** in SQL
* k = similar to **'group** **by'** in SQL

**Ex:**

* DT [c1 > 5, sum(c2), by = c3]
* Fetches all rows with c1 > 5 and then groups by c3 and calculates sum of c2 and returns sum(c2)

1. **Basic commands:**

library(data.table)

dt = data.table(a = runif(100), b = runif(100), c = runif(100), d = sample(letters, 100, replace = T), e = sample(letters, 100, replace = T))

ddt = fread(file.choose())

fwrite(dt, 'file-name.csv')

1. **Check the data.tables in environment**

tables()

1. **Playing with i in DT Syntax:**

dt[a > 0.5]

dt[a > 0.5 & b > 0.5]

1. **Playing with j in DT Syntax:**

dt[, b]

* Returns only 'b' column

dt[, .(b, c)]

* Returns b, c columns
* **.(b, c)** is similar to **list('b', 'c')**

1. **Playing with i, j together in DT Syntax:**

dt[a > 0.5, .(b, c)]

dt[a > 0.5 & d %in% c('a', 'b'), .(b, c)]

1. **Playing with j, k together in DT Syntax:**

dt[, sum(a), by = d]

* same as 'select sum(a) from dt group by d'

dt[, .(sum(a), sum(b)), by = d]

* same as 'select sum(a), sum(b) from dt group by d'

dt[, sum(a), by = .(d, e)]

* same as 'select sum(a) from dt group by d, e'

1. **Filtering using key**
   * ***Note:*** *Twice faster than normal filtering in data.table*

setkey(dt, d)

dt[c('a', 'b')]

* same as dt[d %in% c('a', 'b')]

setkey(dt, d, e)

dt[.('a', 'b')]

* returns a row where d == 'a' and e == 'b'.. i.e. same as **dt [d == 'a' & e == 'b']**

1. **Ordering/Sorting**

setorder(dt, c)

* ascending

setorder(dt , -c)

* descending

setorder(dt, a, -c)

* multiple columns sorting

1. **Adding columns**

dt[, f := runif(100)]

* adds a column

dt[, c('f', 'g') := list(runif(100), runif(100))]

* Adds multiple columns at a time

1. **Summarizing**

dt[, .(mean = mean(a), median = median(a))]

dt[, .(b, c, mean = mean(a))]

* since mean(a) returns only one value it recycles the value of mean(a) and appends it to all the rows.

1. **.SD = Subset of Data**

dt[, lapply(.SD, mean), .SDcols = c("a", "b", "c")]

* apply the mean for subset of columns.

dt[, lapply(.SD, mean)]

* summarizes all columns - in this case it returns NA for d, e columns (as they are character type)

dt[, sapply(.SD, function(x) c(mean=mean(x), median=median(x))), .SDcols = c('a', 'b', 'c')]

* multiple statistics on subset of columns.

1. **.N = Number of rows/Count**

dt[.N]

* fetches last row

dt[, .N]

* fetches number of rows

dt[, .N, by = e]

* same as 'select count(\*) from dt group by e'

1. **Aggregation - very powerful**

dt[, mean(a), by = e]

* default column name for mean(a) will be V1

dt[, .(mean(a), mean(b)), by = d]

* default column names for mean(a), mean(b) will be V1, V2

dt[, .(mean = mean(a)), by = e]

* renames mean(a) from V1 to mean

dt[, .(mean\_a = mean(a), mean\_b = mean(b)), by = d]

* default column names will be V1, V2

dt[, lapply(.SD, mean, na.rm = TRUE), .SDcols = c("a", "b"), by = d]

* renames to a, b directly

1. **Remove duplicates**

setkey(dt, NULL)

* removes set keys

unique(dt)

* removes duplicates in the entire data.table

setkey(dt, e)

* key value = e

unique(dt)

* remove duplicates in e

1. **Extract selected values in group**

dt[, .SD[1:2], by = e]

* selects top two rows for every level in e

dt[, SD[.N], by = e]

* selects last row for every level in e

1. **frank – same as rank function in base R**

dt[, .(rank = frank(a, ties.method = "min")), by = e]

* assigns 1 for the minimum value of a in each level of e, 2 for the 2nd min value ... etc.

dt[, .(rank = frank(-a, ties.method = "min")), by = e]

* same as above but 1 for max value (observe the - sign for a)

1. **like (works more like grep), between**

dt[, a %between% c(0.3, 0.5)]

dt[, e %like% c('a', 'e')]

1. **Joins**

dt1 = data.table(a1 = runif(26), b = sample(letters))

dt2 = data.table(a2 = runif(26), b = sample(letters))

setkey(dt1, b)

setkey(dt2, b)

merge(dt1, dt2, by = 'b')

* uses the merge in data.table instead of base package !! - faster!!

1. **melt, dcast**

* same syntax as dply
* faster in data.table than dplyr package !!

1. **Conversions**

setDF(dt)

* converts data.table to data.frame

setDT(dt)

* converts data.frame to data.table