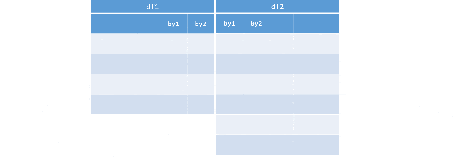
**Introduction**

In this post in the R:case4base series we will look at one of the most common operations on multiple data frames – merge, also known as JOIN in SQL terms.

We will learn how to do the 4 basic types of join – inner, left, right and full join with base R and show how to perform the same with tidyverse’s dplyr and data.table’s methods. A quick benchmark will also be included.



Joins

**Merging (joining) two data frames with base R**

To showcase the merging, we will use a very slightly modified dataset provided by nycflights13 package, mainly the flights and weather data frames. Let’s get right into it and simply show how to perform the different types of joins with base R.

First, we prepare the data and store the columns we will merge by (join on) into mergeCols:

dataurl <- "<https://jozefhajnala.gitlab.io/r/post/data/>"

weather <- readRDS(url(paste0(dataurl, "r006/weather.rds")))

flights <- readRDS(url(paste0(dataurl, "r006/flights.rds")))

mergeCols <- c("time\_hour", "origin")

head(flights)

## year month day dep\_time sched\_dep\_time dep\_delay arr\_time sched\_arr\_time

## 1 2013 1 1 517 515 2 830 819

## 2 2013 1 1 533 529 4 850 830

## 3 2013 1 1 542 540 2 923 850

## 4 2013 1 1 544 545 -1 1004 1022

## 5 2013 1 1 554 600 -6 812 837

## 6 2013 1 1 554 558 -4 740 728

## arr\_delay carrier flight tailnum origin dest air\_time distance hour

## 1 11 UA 1545 N14228 EWR IAH 227 1400 5

## 2 20 UA 1714 N24211 LGA IAH 227 1416 5

## 3 33 AA 1141 N619AA JFK MIA 160 1089 5

## 4 -18 B6 725 N804JB JFK BQN 183 1576 5

## 5 -25 DL 461 N668DN LGA ATL 116 762 6

## 6 12 UA 1696 N39463 EWR ORD 150 719 5

## minute time\_hour

## 1 15 2013-01-01 05:00:00

## 2 29 2013-01-01 05:00:00

## 3 40 2013-01-01 05:00:00

## 4 45 2013-01-01 05:00:00

## 5 0 2013-01-01 06:00:00

## 6 58 2013-01-01 05:00:00

head(weather)

## origin year month day hour temp dewp humid wind\_dir wind\_speed

## 1 EWR 2013 1 1 1 39.02 26.06 59.37 270 10.35702

## 2 EWR 2013 1 1 2 39.02 26.96 61.63 250 8.05546

## 3 EWR 2013 1 1 3 39.02 28.04 64.43 240 11.50780

## 4 EWR 2013 1 1 4 39.92 28.04 62.21 250 12.65858

## 5 EWR 2013 1 1 5 39.02 28.04 64.43 260 12.65858

## 6 EWR 2013 1 1 6 37.94 28.04 67.21 240 11.50780

## wind\_gust precip pressure visib time\_hour

## 1 NA 0 1012.0 10 2013-01-01 01:00:00

## 2 NA 0 1012.3 10 2013-01-01 02:00:00

## 3 NA 0 1012.5 10 2013-01-01 03:00:00

## 4 NA 0 1012.2 10 2013-01-01 04:00:00

## 5 NA 0 1011.9 10 2013-01-01 05:00:00

## 6 NA 0 1012.4 10 2013-01-01 06:00:00

Now, we show how to perform the 4 merges (joins):

**Inner join**

inner <- merge(flights, weather, by = mergeCols)

**Left (outer) join**

left <- merge(flights, weather, by = mergeCols, all.x = TRUE)

**Right (outer) join**

right <- merge(flights, weather, by = mergeCols, all.y = TRUE)

**Full (outer) join**

full <- merge(flights, weather, by = mergeCols, all = TRUE)

**Other join types**

# Cross Join (Cartesian product)

cross <- merge(flights, weather, by = NULL)

# Natural Join

natural <- merge(flights, weather)

**The arguments of merge**

The key arguments of base merge data.frame method are:

* x, y – the 2 data frames to be merged
* by – names of the columns to merge on. If the column names are different in the two data frames to merge, we can specify by.x and by.y with the names of the columns in the respective data frames. The by argument can also be specified by number, logical vector or left unspecified, in which case it defaults to the intersection of the names of the two data frames. From best practice perspective it is advisable to always specify the argument explicitly, ideally by column names.
* all, all.x, all.y – default to FALSE and can be used specify the type of join we want to perform:
  + all = FALSE (the default) – gives an inner join – combines the rows in the two data frames that match on the by columns
  + all.x = TRUE – gives a left (outer) join – adds rows that are present in x, even though they do not have a matching row in y to the result for all = FALSE
  + all.y = TRUE – gives a right (outer) join – adds rows that are present in y, even though they do not have a matching row in x to the result for all = FALSE
  + all = TRUE – gives a full (outer) join. This is a shorthand for all.x = TRUE and all.y = TRUE

Other arguments include

* sort – if TRUE (default), results are sorted on the by columns
* suffixes – length 2 character vector, specifying the suffixes to be used for making the names of columns in the result which are not used for merging unique
* incomparables – for single-column merging only, a vector of values that cannot be matched. Any value in x matching a value in this vector is assigned the nomatch value (which can be passed using ...)

**Merging multiple data frames**

For this example, let us have a list of all the data frames included in the nycflights13 package, slightly updated such that they can me merged with the default value for by, purely for this exercise, and store them into a list called flightsList:

flightsList <- readRDS(url(paste0(dataurl, "r006/nycflights13-list.rds")))

lapply(flightsList, function(x) c(toString(dim(x)), toString(names(x))))

## $flights

## [1] "336776, 19"

## [2] "year, month, day, dep\_time, sched\_dep\_time, dep\_delay, arr\_time, sched\_arr\_time, arr\_delay, carrier, flight, tailnum, origin, dest, air\_time, distance, hour, minute, time\_hour"

##

## $weather

## [1] "26115, 15"

## [2] "origin, year, month, day, hour, temp, dewp, humid, wind\_dir, wind\_speed, wind\_gust, precip, pressure, visib, time\_hour"

##

## $airlines

## [1] "16, 2" "carrier, name"

##

## $airports

## [1] "1458, 8"

## [2] "origin, airportname, lat, lon, alt, tz, dst, tzone"

##

## $planes

## [1] "3322, 9"

## [2] "tailnum, yearmanufactured, type, manufacturer, model, engines, seats, speed, engine"

Since merge is designed to work with 2 data frames, merging multiple data frames can of course be achieved by nesting the calls to merge:

multiFull <- merge(merge(merge(merge(

flightsList[[1L]],

flightsList[[2L]], all = TRUE),

flightsList[[3L]], all = TRUE),

flightsList[[4L]], all = TRUE),

flightsList[[5L]], all = TRUE)

We can however achieve this same goal much more elegantly, taking advantage of base R’s Reduce function:

# For Inner Join

multi\_inner <- Reduce(

function(x, y, ...) merge(x, y, ...),

flightsList

)

# For Full (Outer) Join

multi\_full <- Reduce(

function(x, y, ...) merge(x, y, all = TRUE, ...),

flightsList

)

Note that this example is oversimplified and the data was updated such that the default values for by give meaningful joins. For example, in the original planes data frame the column year would have been matched onto the year column of the flights data frame, which is nonsensical as the years have different meanings in the two data frames. This is why we renamed the year column in the planes data frame to yearmanufactured for the above example.

**Alternatives to base R**

**Using the tidyverse**

The dplyr package comes with a set of very user-friendly functions that seem quite self-explanatory:

library(dplyr)

inner\_dplyr <- inner\_join(flights, weather, by = mergeCols)

left\_dplyr <- left\_join(flights, weather, by = mergeCols)

right\_dplyr <- right\_join(flights, weather, by = mergeCols)

full\_dplyr <- full\_join(flights, weather, by = mergeCols)

We can also use the “forward pipe” operator %>% that becomes very convenient when merging multiple data frames:

inner\_dplyr <- flights %>% inner\_join(weather, by = mergeCols)

left\_dplyr <- flights %>% left\_join(weather, by = mergeCols)

right\_dplyr <- flights %>% right\_join(weather, by = mergeCols)

full\_dplyr <- flights %>% full\_join(weather, by = mergeCols)

**Using data.table**

The data.table package provides an S3 method for the merge generic that has a very similar structure to the base method for data frames, meaning its use is very convenient for those familiar with that method. In fact the code is exactly the same as the base one for our example use.

One important difference worth noting is that the by argument is by default constructed differently with data.table.

We however provide it explicitly, therefore this difference does not directly affect our example:

library(data.table)

weather <- as.data.table(weather)

flights <- as.data.table(flights)

setkeyv(weather, mergeCols)

setkeyv(flights, mergeCols)

# Note that this is identical to the code for base

# The data.table method is called automatically for objects of class data.table

inner\_dt <- merge(flights, weather, by = mergeCols)

left\_dt <- merge(flights, weather, by = mergeCols, all.x = TRUE)

right\_dt <- merge(flights, weather, by = mergeCols, all.y = TRUE)

full\_dt <- merge(flights, weather, by = mergeCols, all = TRUE)

Alternatively, we can write data.table joins as subsets:

inner\_dt <- flights[weather, on = mergeCols, nomatch = 0]

left\_dt <- weather[flights, on = mergeCols]

right\_dt <- flights[weather, on = mergeCols]

**Quick benchmarking**

For a quick overview, lets look at a basic benchmark without package loading overhead for each of the mentioned packages:

**Inner join**

bench\_inner <- microbenchmark::microbenchmark(times = 100,

base = base::merge.data.frame(flights, weather, by = mergeCols),

base\_nosort = base::merge.data.frame(flights, weather, by = mergeCols, sort = FALSE),

dt\_merge = merge(flights, weather, by = mergeCols),

dt\_subset = flights[weather, on = mergeCols, nomatch = 0],

dplyr = inner\_join(flights, weather, by = mergeCols),

dplyr\_pipe = flights %>% inner\_join(weather, by = mergeCols)

)

**Full (outer) join**

bench\_outer <- microbenchmark::microbenchmark(times = 100,

base = base::merge.data.frame(flights, weather, by = mergeCols, all = TRUE),

base\_nosort = base::merge.data.frame(flights, weather, by = mergeCols, all = TRUE, sort = FALSE),

dt\_merge = merge(flights, weather, by = mergeCols, all = TRUE),

dplyr = full\_join(flights, weather, by = mergeCols),

dplyr\_pipe = flights %>% full\_join(weather, by = mergeCols)

)

Visualizing the results in this case shows base R comes way behind the two alternatives, even with sort = FALSE.