Merges and Joins

MSDS 597 Data Wrangling & Husbandry

2/10/2020

A Quick Aside on Set Operations

I find the functions `intersect()`, `union()`, and `setdiff()` useful, especially for debugging datasets (e.g., which IDs are in one dataset but not the other).

```
x <- c("A", "B", "C")
y <- c("C", "D")
intersect(x, y)
```

```
## [1] "C"
```

```
union(x,y)
```

```
## [1] "A" "B" "C" "D"
```

```r
x <- c("A", "B", "C")
y <- c("C", "D")

setdiff(x, y)
```

```
## [1] "A" "B"
```

```r
setdiff(y, x)
```

```
## [1] "D"
```

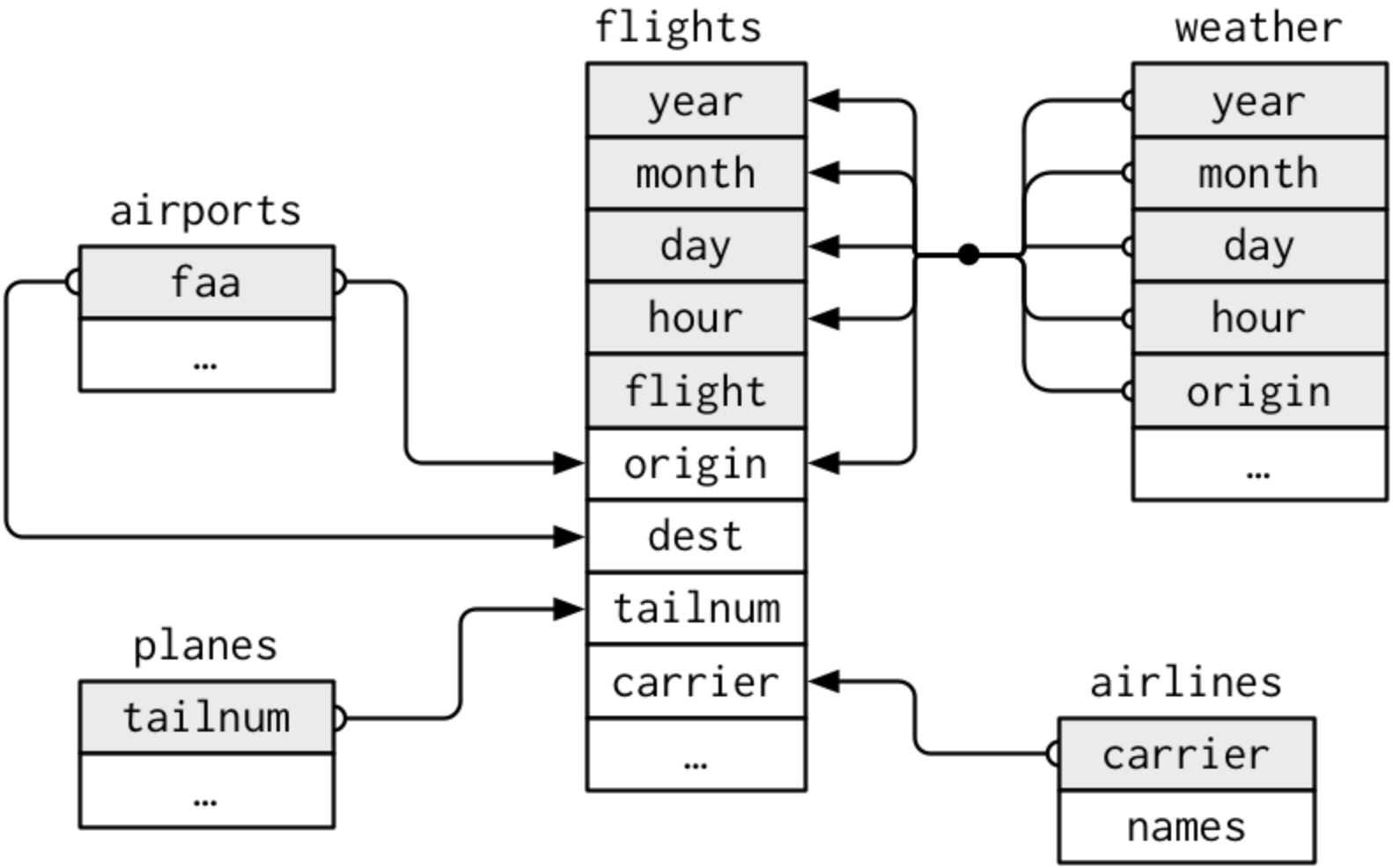
Merge (base R)

Base R has a `merge()` function to merge to data frames

```
merge(x, y, by = intersect(names(x), names(y)),
      by.x = by, by.y = by, all = FALSE, all.x = all, all.y = all,
      sort = TRUE, suffixes = c(".x",".y"),
      incomparables = NULL, ...)
```

Joins

- The `dplyr` package's `join` functions are essentially faster versions of merge, with syntax like SQL. (The following material, including all figures, draws heavily from R for Data Science .)
- Although our fundamental way to store data is in a data frame, the data in data frames is often related to data in other data frames—we might say the data is *relational*.


flights
year
month
day
hour
flight
origin
dest
tailnum
carrier
…
weather
year
month
day
hour
origin
…
airports
faa
…
planes
tailnum
…
airlines
carrier
names

Keys

The variables that connect two tables are known as *keys*. They uniquely identify an observation, but it may require several variables to make the observation unique. There are

- *Primary keys* that uniquely identify an observation in its own table (examples in the preceding figure?)
- *Foreign keys* that identify observations in another table
- *Surrogate keys* that can be created when a table lacks a primary key
  - Can be created with `mutate()` and `row_number()`

```
library(Lahman)
glimpse(Batting)
```

```
## Observations: 105,861
## Variables: 22
## $ playerID <chr> "abercda01", "addybo01", "allisar01", "allisdo01", "ansonca0…
## $ yearID   <int> 1871, 1871, 1871, 1871, 1871, 1871, 1871, 1871, 1871, 1871, …
## $ stint    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
## $ teamID   <fct> TRO, RC1, CL1, WS3, RC1, FW1, RC1, BS1, FW1, BS1, CL1, CL1, …
## $ lgID     <fct> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, …
## $ G        <int> 1, 25, 29, 27, 25, 12, 1, 31, 1, 18, 22, 1, 10, 3, 20, 29, 1…
## $ AB       <int> 4, 118, 137, 133, 120, 49, 4, 157, 5, 86, 89, 3, 36, 15, 94,…
## $ R        <int> 0, 30, 28, 28, 29, 9, 0, 66, 1, 13, 18, 0, 6, 7, 24, 26, 0, …
## $ H        <int> 0, 32, 40, 44, 39, 11, 1, 63, 1, 13, 27, 0, 7, 6, 33, 32, 0,…
## $ X2B      <int> 0, 6, 4, 10, 11, 2, 0, 10, 1, 2, 1, 0, 0, 0, 9, 3, 0, 0, 1, …
## $ X3B      <int> 0, 0, 5, 2, 3, 1, 0, 9, 0, 1, 10, 0, 0, 0, 1, 3, 0, 0, 1, 0,…
## $ HR       <int> 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 1, 0, 0, 0, 0, 0, …
## $ RBI      <int> 0, 13, 19, 27, 16, 5, 2, 34, 1, 11, 18, 0, 1, 5, 21, 23, 0, …
## $ SB       <int> 0, 8, 3, 1, 6, 0, 0, 11, 0, 1, 0, 0, 2, 2, 4, 4, 0, 0, 3, 0,…
## $ CS       <int> 0, 1, 1, 1, 2, 1, 0, 6, 0, 0, 1, 0, 0, 0, 0, 4, 0, 0, 1, 0, …
## $ BB       <int> 0, 4, 2, 0, 2, 0, 1, 13, 0, 0, 3, 1, 2, 0, 2, 9, 0, 0, 4, 1,…
## $ SO       <int> 0, 0, 5, 2, 1, 1, 0, 1, 0, 0, 4, 0, 0, 0, 2, 2, 3, 0, 2, 0, …
## $ IBB      <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, …
## $ HBP      <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, …
## $ SH       <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, …
```

```
names(Batting)
```

```
##  [1] "playerID" "yearID"   "stint"    "teamID"   "lgID"     "G"
##  [7] "AB"       "R"        "H"        "X2B"      "X3B"      "HR"
## [13] "RBI"      "SB"       "CS"       "BB"       "SO"       "IBB"
## [19] "HBP"      "SH"       "SF"       "GIDP"
```

```
names(Master)
```

```
##  [1] "playerID"     "birthYear"    "birthMonth"   "birthDay"     "birthCountry"
##  [6] "birthState"   "birthCity"    "deathYear"    "deathMonth"   "deathDay"
## [11] "deathCountry" "deathState"   "deathCity"    "nameFirst"    "nameLast"
## [16] "nameGiven"    "weight"       "height"       "bats"         "throws"
## [21] "debut"        "finalGame"    "retroID"      "bbrefID"      "deathDate"
## [26] "birthDate"
```

```
names(Teams)
```

```
##  [1] "yearID"         "lgID"           "teamID"         "franchID"
##  [5] "divID"          "Rank"           "G"              "Ghome"
##  [9] "W"              "L"              "DivWin"         "WCWin"
## [13] "LgWin"          "WSWin"          "R"              "AB"
## [17] "H"              "X2B"            "X3B"            "HR"
## [21] "BB"             "SO"             "SB"             "CS"
## [25] "HBP"            "SF"             "RA"             "ER"
## [29] "ERA"            "CG"             "SHO"            "SV"
## [33] "IPouts"         "HA"             "HRA"            "BBA"
## [37] "SOA"            "E"              "DP"             "FP"
## [41] "name"           "park"           "attendance"     "BPF"
## [45] "PPF"            "teamIDBR"       "teamIDlahman45" "teamIDretro"
```

By the way, the Lahman package has a convenient function `Label()`

```
Label("lgID")
```

```
## [1] "League"
```

It's always worth checking whether a primary key is in fact unique

```
Batting %>% count(playerID) %>% filter(n > 1)
```

```
## # A tibble: 14,327 x 2
##    playerID      n
##    <chr>     <int>
##  1 aardsda01     9
##  2 aaronha01    23
##  3 aaronto01     7
##  4 aasedo01     13
##  5 abadan01      3
##  6 abadfe01      9
##  7 abadijo01     2
##  8 abbated01    10
##  9 abbeybe01     6
## 10 abbeych01     5
## # … with 14,317 more rows
```

So `playerID` is not a primary key for the `Batting` data frame.

```
Batting %>% count(playerID, yearID) %>% filter(n > 1)
```

```
## # A tibble: 7,432 x 3
##    playerID  yearID     n
##    <chr>      <int> <int>
##  1 abadfe01    2016     2
##  2 abadijo01   1875     2
##  3 abbated01   1910     2
##  4 abbeybe01   1895     2
##  5 abbotgl01   1983     2
##  6 abbotji01   1995     2
##  7 abbotku01   1998     2
##  8 abbotpa01   2004     2
##  9 aberal01    1953     2
## 10 aberal01    1957     2
## # … with 7,422 more rows
```

Nope

```
Batting %>% count(playerID, yearID, teamID) %>% filter(n > 1)
```

```
## # A tibble: 69 x 4
##    playerID  yearID teamID     n
##    <chr>      <int> <fct>  <int>
##  1 alyeabr01   1972 OAK        2
##  2 anderjo01   1898 BRO        2
##  3 baldwja01   2005 BAL        2
##  4 behrmha01   1947 BRO        2
##  5 chouife01   1914 BRF        3
##  6 clarkje02   2003 TEX        2
##  7 clarkni01   1905 CLE        2
##  8 cranddo01   1913 NY1        2
##  9 cranesa01   1890 NY1        2
## 10 donahpa01   1910 PHA        2
## # … with 59 more rows
```

```
filter(Batting, playerID == "baldwja01" & yearID == 2005)
```

```
##    playerID yearID stint teamID lgID  G AB R H X2B X3B HR RBI SB CS BB SO IBB
## 1 baldwja01   2005     1    BAL   AL 12  1 0 0   0   0  0   0  0  0  0  0   0
## 2 baldwja01   2005     2    TEX   AL  8  0 0 0   0   0  0   0  0  0  0  0   0
## 3 baldwja01   2005     3    BAL   AL  8  0 0 0   0   0  0   0  0  0  0  0   0
##   HBP SH SF GIDP
## 1   0  0  0    0
## 2   0  0  0    0
## 3   0  0  0    0
```

```
Batting %>% count(playerID, yearID, stint) %>% filter(n > 1)
```

```
## # A tibble: 0 x 4
## # … with 4 variables: playerID <chr>, yearID <int>, stint <int>, n <int>
```

Finally.

- So `playerID`, `yearID`, `stint` form a primary key for the `Batting` data frame.
- However, in the `Master` data frame, `playerID` alone is a primary key

```
Master %>% count(playerID) %>% filter(n > 1)
```

```
## # A tibble: 0 x 2
## # … with 2 variables: playerID <chr>, n <int>
```

- `Batting$playerID` is a foreign key, since it is a primary key for the `Master` data frame and matches each observation in `Batting` to a unique player.

Mutating joins

- A mutating join creates a new data frame from two data frames (two “tables” in database jargon) by combining variables based on the keys.
- Think of it as
  1. match by keys
  2. copy variables from one data frame to other
- As an example we start with `left_join(x, y)` (an example of an “outer join”)
  - left_join(x, y) returns all rows from x, and all columns from x and y. Rows in x with no match in y will have NA values in the new columns. If there are multiple matches between x and y, all combinations of the matches are returned.

```
Batting.small <- Batting %>%
  select(playerID, yearID, stint, teamID, lgID, G, AB)
Master.small <- Master %>%
  select(nameFirst, nameLast, playerID)
left_join(Batting.small, Master.small) %>% head()
```

```
## Joining, by = "playerID"
```

```
##    playerID yearID stint teamID lgID  G  AB nameFirst    nameLast
## 1 abercda01   1871     1    TRO   NA  1   4     Frank Abercrombie
## 2  addybo01   1871     1    RC1   NA 25 118       Bob        Addy
## 3 allisar01   1871     1    CL1   NA 29 137       Art     Allison
## 4 allisdo01   1871     1    WS3   NA 27 133      Doug     Allison
## 5 ansonca01   1871     1    RC1   NA 25 120       Cap       Anson
## 6 armstbo01   1871     1    FW1   NA 12  49    Robert   Armstrong
```

```
left_join(Batting.small, Master.small) %>%
  filter(playerID == "ruthba01")
```

```
## Joining, by = "playerID"
```

```
##    playerID yearID stint teamID lgID   G  AB nameFirst nameLast
## 1  ruthba01   1914     1    BOS   AL   5  10      Babe     Ruth
## 2  ruthba01   1915     1    BOS   AL  42  92      Babe     Ruth
## 3  ruthba01   1916     1    BOS   AL  67 136      Babe     Ruth
## 4  ruthba01   1917     1    BOS   AL  52 123      Babe     Ruth
## 5  ruthba01   1918     1    BOS   AL  95 317      Babe     Ruth
## 6  ruthba01   1919     1    BOS   AL 130 432      Babe     Ruth
## 7  ruthba01   1920     1    NYA   AL 142 457      Babe     Ruth
## 8  ruthba01   1921     1    NYA   AL 152 540      Babe     Ruth
## 9  ruthba01   1922     1    NYA   AL 110 406      Babe     Ruth
## 10 ruthba01   1923     1    NYA   AL 152 522      Babe     Ruth
## 11 ruthba01   1924     1    NYA   AL 153 529      Babe     Ruth
## 12 ruthba01   1925     1    NYA   AL  98 359      Babe     Ruth
## 13 ruthba01   1926     1    NYA   AL 152 495      Babe     Ruth
## 14 ruthba01   1927     1    NYA   AL 151 540      Babe     Ruth
## 15 ruthba01   1928     1    NYA   AL 154 536      Babe     Ruth
## 16 ruthba01   1929     1    NYA   AL 135 499      Babe     Ruth
## 17 ruthba01   1930     1    NYA   AL 145 518      Babe     Ruth
## 18 ruthba01   1931     1    NYA   AL 145 534      Babe     Ruth
```

The form of all of the join commands is like

```
left_join(x, y, by = NULL, suffix = c(".x", ".y"))
```

- where `x` is a data frame, `y` is a data frame, `by` is a character vector of variable to join on (the default is all names in the intersection), and `suffix` is what to add to duplicate variable sthat aren't part of the key.
- Only the first two slots are required.
- With pipes, you would write

```
x %>% left_join(y)
```

Visualizing joins

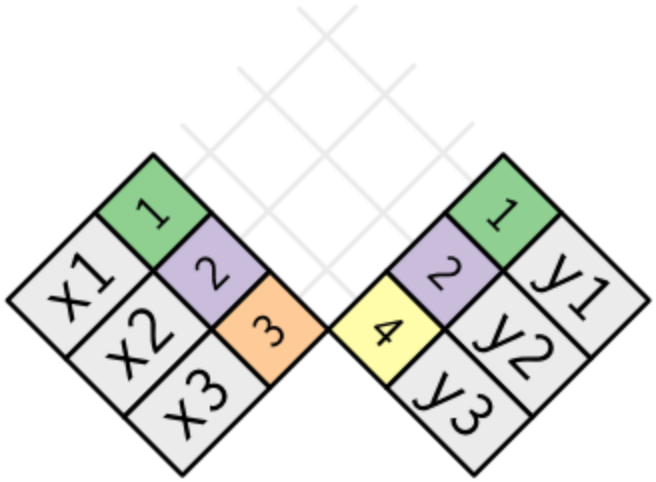
x

1	x1
2	x2
3	x3

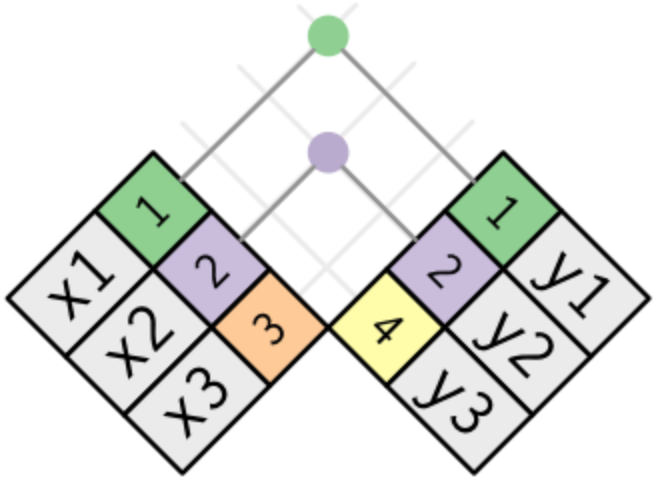
y

1	y1
2	y2
4	y3

The figure below shows all the ways we might match observations in the two tables.



Inner join



key	val_x	val_y
1	x1	y1
2	x2	y2

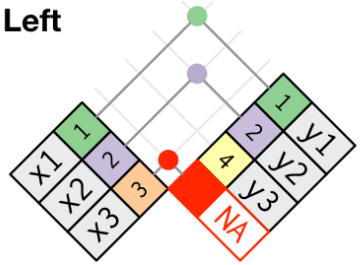
- An inner join matches pairs of observations only when their keys are equal.
- `inner_join(x, y, by = "key")`
- **Unmatched observations are dropped!**
- equivalent to `merge(x, y, by = "key", all = FALSE)`

Outer joins

We typically instead want “outer joins.”

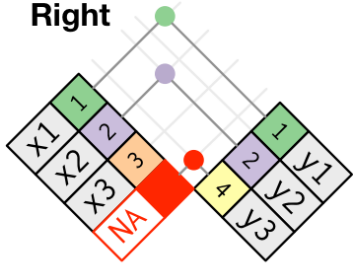
- Left joins keep all the observations in left hand dataset
  - `left_join(x, y, by = “key”)
  - `merge(x, y, by = “key”, all.x = TRUE)
- Right joins keep all the observations in the right hand dataset
  - `right_join(x, y, by = “key”)
  - `merge(x, y, by = “key”, all.y = TRUE)
- Full joins keep all the observations in both
  - `full_join(x, y, by = “key”)
  - `merge(x, y, by = “key”, all = TRUE)

Left



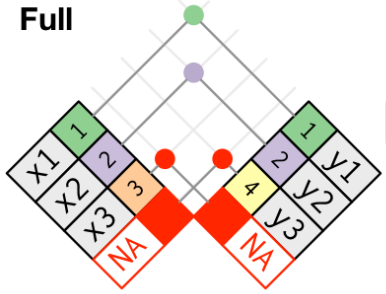
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Right



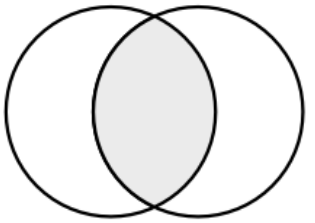
key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

Full

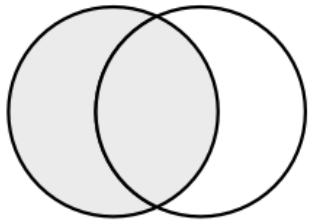


key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

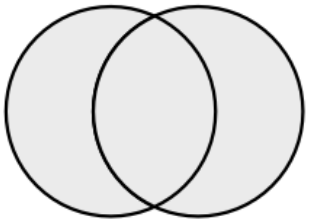
These Venn diagrams might help with the names.



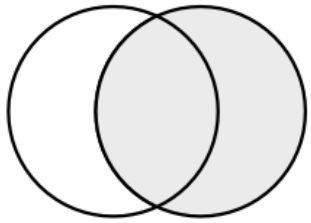
`inner_join(x, y)`



`left_join(x, y)`



`full_join(x, y)`

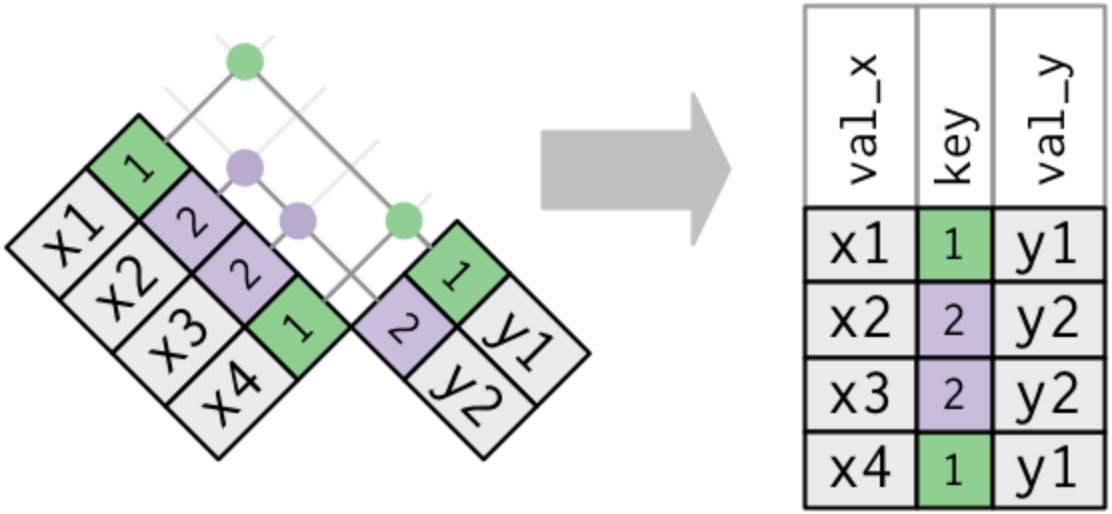


`right_join(x, y)`

More on keys in the join functions

- If you don't include the `by` argument, the key will be all variables that appear in both tables (called a "natural join").
- To use several variables for the key, just use a vector of names for the `by` argument
- To use different variables for the key in the two tables, use a named vector
  - `by = c("subjectID" = "ID", "yr" = "year")` will match `subjectID` and `yr` in the left table to `ID` and `year` in the right table.
- When the keys are not unique, the situation is a bit more complicated . . .

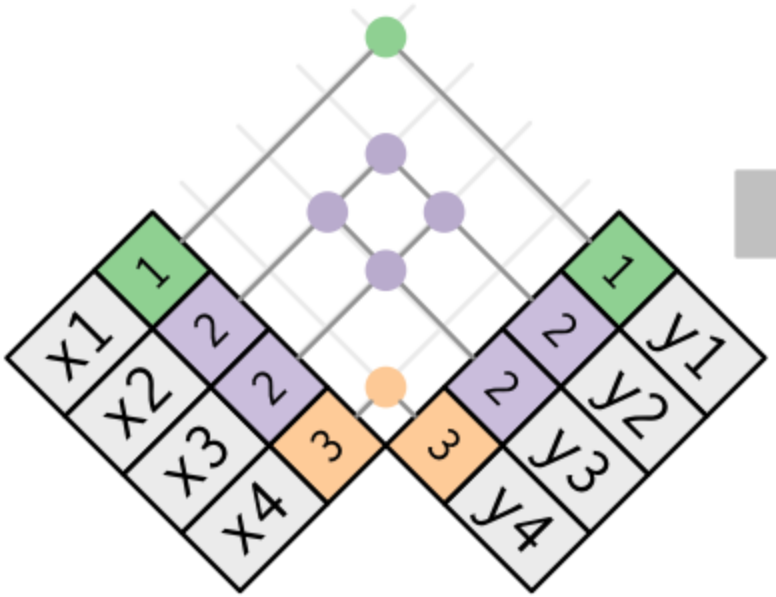
When one table has duplicate keys



val_x	key	val_y
x1	1	y1
x2	2	y2
x3	2	y2
x4	1	y1

- This is a typical many-to-one relationship.
- `left_join(Batting.small, Master.small)` was an example of this
  - each `playerID` appeared many times in `Batting.small` but just once in `Master.small`

When both tables have duplicate keys



key	val_x	val_y
1	x1	y1
2	x2	y2
2	x2	y3
2	x3	y2
2	x3	y3
3	x4	y4

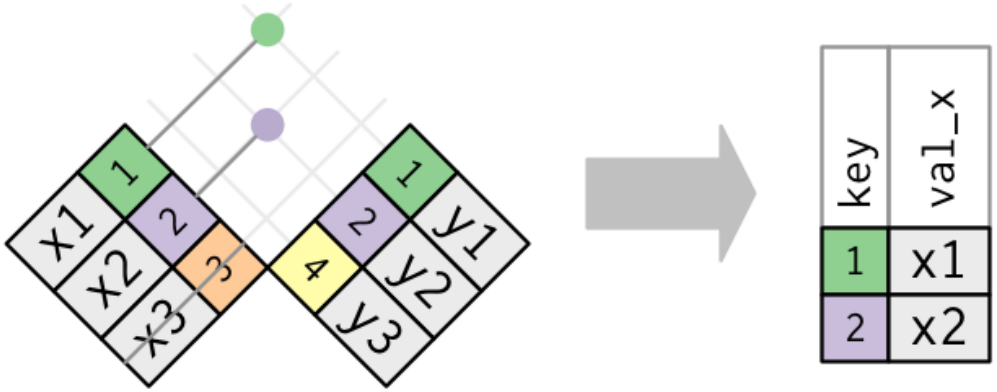
This is usually a mistake.

Filtering joins

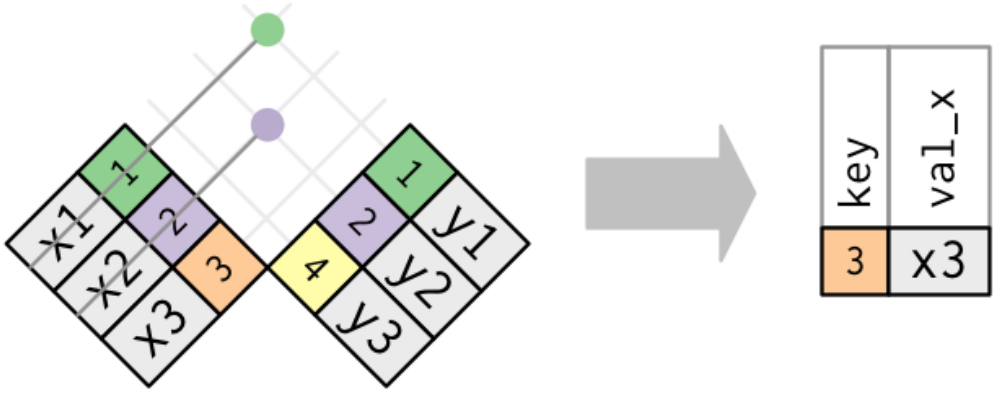
Filtering joints extract cases from one data set using information from another.

- `semi_join()` will return rows in `x` where the keys match, but keeping just the columns from `x` and so not returning duplicate rows of `x`.
- `anti_join()` will return rows where the keys do not match
- These are useful to debug problems (e.g., typos in keys) and to otherwise understand what happens after a join.

Semi-join:



Anti-join:



Example of an anti-join

```
library(nycflights13)
glimpse(airports)
```

```
## Observations: 1,458
## Variables: 8
## $ faa   <chr> "04G", "06A", "06C", "06N", "09J", "0A9", "0G6", "0G7", "0P2", …
## $ name  <chr> "Lansdowne Airport", "Moton Field Municipal Airport", "Schaumbu…
## $ lat   <dbl> 41.13047, 32.46057, 41.98934, 41.43191, 31.07447, 36.37122, 41.…
## $ lon   <dbl> -80.61958, -85.68003, -88.10124, -74.39156, -81.42778, -82.1734…
## $ alt   <dbl> 1044, 264, 801, 523, 11, 1593, 730, 492, 1000, 108, 409, 875, 1…
## $ tz    <dbl> -5, -6, -6, -5, -5, -5, -5, -5, -5, -8, -5, -6, -5, -5, -5, -5,…
## $ dst   <chr> "A", "A", "A", "A", "A", "A", "A", "A", "U", "A", "A", "U", "A"…
## $ tzone <chr> "America/New_York", "America/Chicago", "America/Chicago", "Amer…
```

```
glimpse(flights)
```

```
## Observations: 336,776
## Variables: 19
## $ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, …
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
## $ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558,…
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600,…
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -…
## $ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849…
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851…
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -…
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", …
## $ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, …
## $ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N39…
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA"…
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD"…
## $ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, …
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733,…
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, …
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, …
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 …
```

```
## Note how 'by' is used wnen the variable names are different
anti_join(airports, flights, by = c("faa" = "dest"))
```

```
## # A tibble: 1,357 x 8
##    faa   name                           lat    lon   alt    tz dst   tzone
##    <chr> <chr>                        <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
##  1 04G   Lansdowne Airport             41.1  -80.6  1044    -5 A     America/New_Yo…
##  2 06A   Moton Field Municipal A…      32.5  -85.7   264    -6 A     America/Chicago
##  3 06C   Schaumburg Regional           42.0  -88.1   801    -6 A     America/Chicago
##  4 06N   Randall Airport               41.4  -74.4   523    -5 A     America/New_Yo…
##  5 09J   Jekyll Island Airport         31.1  -81.4    11    -5 A     America/New_Yo…
##  6 0A9   Elizabethton Municipal …      36.4  -82.2  1593    -5 A     America/New_Yo…
##  7 0G6   Williams County Airport       41.5  -84.5   730    -5 A     America/New_Yo…
##  8 0G7   Finger Lakes Regional A…      42.9  -76.8   492    -5 A     America/New_Yo…
##  9 0P2   Shoestring Aviation Air…      39.8  -76.6  1000    -5 U     America/New_Yo…
## 10 0S9   Jefferson County Intl         48.1 -123.    108    -8 A     America/Los_An…
## # … with 1,347 more rows
```

In class exercise:

- First exercise
  - Create a data frame from the file “gini.txt” in the class folder on Canvas. *You will have to add column names.*
  - Merge that data frame with the 2007 gapminder data frame.
  - Determine which countries are in one data frame but not the other by using the `setdiff()` as well as by the `anti_join()` function.
- Second exercise
  - Using the `Latham` package, add columns to the SeriesPost data frame where the winning team (`teamIDwinner`) is expanded to the team name (`name` in the `Teams` data frame). Note that team names can change by year.
  - Use `anti_join` to diagnose problems.