R Lab4: Data Management(II)

QSSBA,2025

Zhaopeng Qu

HNC

March 24 2025

1 Data Management in R: Relational data and Reshape data

2 Introduction to Reshape data

Section 1

Data Management in R: Relational data and Reshape data

Introduction to Relational data

- Collectively, multiple tables of data are called relational data because it is the relations, not just the individual datasets, that are important.
- eg. a household survey often composes with several subsets:
  - some variables are reported in **household level**: such as consumption,assets,housing,etc.
  - others are reported in **individual level**: like working information and demographic information.

Types of join

- **Mutation joins** first matches observations by their keys, then copies across variables from one table to the other.
- **Filtering joins** match observations in the same way as mutating joins, but affect the **observations**, not the variables.

nycflights13 data

- a dataset on flights departing New York City in 2013.

```
install.packages("nycflights13")
```

```
library(nycflights13)
airlines
```

```
## # A tibble: 16 x 2
##    carrier name
##    <chr>   <chr>
##  1 9E      Endeavor Air Inc.
##  2 AA      American Airlines Inc.
##  3 AS      Alaska Airlines Inc.
##  4 B6      JetBlue Airways
##  5 DL      Delta Air Lines Inc.
##  6 EV      ExpressJet Airlines Inc.
##  7 F9      Frontier Airlines Inc.
```

nycflights13:airports

```
airports
```

```
## # A tibble: 1,458 x 8
##    faa   name                              lat    lon   alt
##    <chr> <chr>                           <dbl>  <dbl> <dbl>
##  1 04G   Lansdowne Airport                41.1  -80.6  1044
##  2 06A   Moton Field Municipal Airport    32.5  -85.7   264
##  3 06C   Schaumburg Regional              42.0  -88.1   801
##  4 06N   Randall Airport                  41.4  -74.4   523
##  5 09J   Jekyll Island Airport            31.1  -81.4    11
##  6 0A9   Elizabethton Municipal Airport   36.4  -82.2  1593
##  7 0G6   Williams County Airport          41.5  -84.5   730
##  8 0G7   Finger Lakes Regional Airport    42.9  -76.8   492
##  9 0P2   Shoestring Aviation Airfield     39.8  -76.6  1000
## 10 0S9   Jefferson County Intl            48.1 -123.    108
## # i 1,448 more rows
```

nycflights13:planes

```
planes
```

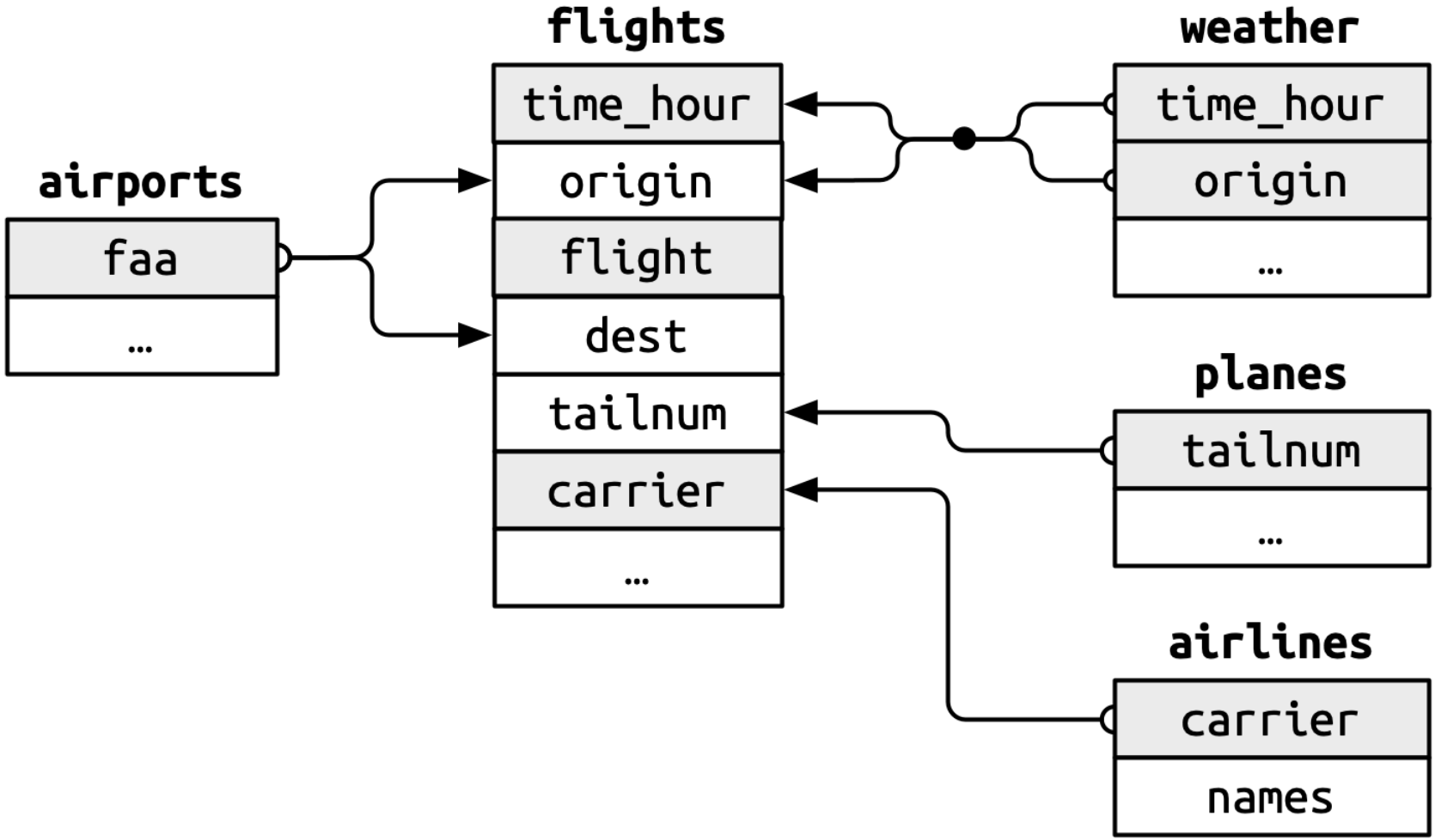
```
## # A tibble: 3,322 x 9
##    tailnum  year type                manufacturer model engi
##    <chr>   <int> <chr>               <chr>        <chr>   <i
##  1 N10156   2004 Fixed wing multi~ EMBRAER      EMB-~
##  2 N102UW   1998 Fixed wing multi~ AIRBUS INDU~ A320~
##  3 N103US   1999 Fixed wing multi~ AIRBUS INDU~ A320~
##  4 N104UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~
##  5 N10575   2002 Fixed wing multi~ EMBRAER      EMB-~
##  6 N105UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~
##  7 N107US   1999 Fixed wing multi~ AIRBUS INDU~ A320~
##  8 N108UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~
##  9 N109UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~
## 10 N110UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~
## # i 3,312 more rows
```

nycflights13:weather

```
weather
```

```
## # A tibble: 26,115 x 15
##    origin  year month   day  hour  temp  dewp humid wind_di
##    <chr>  <int> <int> <int> <int> <dbl> <dbl> <dbl>    <dbl
##  1 EWR     2013     1     1     1  39.0  26.1  59.4       27
##  2 EWR     2013     1     1     2  39.0  27.0  61.6       25
##  3 EWR     2013     1     1     3  39.0  28.0  64.4       24
##  4 EWR     2013     1     1     4  39.9  28.0  62.2       25
##  5 EWR     2013     1     1     5  39.0  28.0  64.4       26
##  6 EWR     2013     1     1     6  37.9  28.0  67.2       24
##  7 EWR     2013     1     1     7  39.0  28.0  64.4       24
##  8 EWR     2013     1     1     8  39.9  28.0  62.2       25
##  9 EWR     2013     1     1     9  39.9  28.0  62.2       26
## 10 EWR     2013     1     1    10  41    28.0  59.6       26
## # i 26,105 more rows
## # i 5 more variables: wind_gust <dbl>, precip <dbl>, pressu
```

Relations in nycflights13 data



Key variables

- The variables used to connect each pair of tables are called **keys**. A key is a variable (or set of variables) that uniquely identifies an observation.
  - **Primary** Key: uniquely identifies an observation in its **own** table
  - **Foreign** Key: uniquely identifies an observation in **another** table

```
head(planes$tailnum) # primary key in `planes` table
```

```
## [1] "N10156" "N102UW" "N103US" "N104UW" "N10575" "N105UW"
```

```
head(flights$tailnum) # foreign key in `flights` table
```

```
## [1] "N14228" "N24211" "N619AA" "N804JB" "N668DN" "N39463"
```

Key variables

- Verify that key variables do indeed uniquely identify each observation.

```
planes %>%
  count(tailnum) %>% # Count occurrences of each tailnum valu
  filter(n > 1)      # Filter to keep only rows where count i
```

```
## # A tibble: 0 x 2
## # i 2 variables: tailnum <chr>, n <int>
```

- or just use distinct() or duplicated()

```
planes %>%
  filter(duplicated(tailnum)) # filter(duplicated(tailnum)) -
```

```
## # A tibble: 0 x 9
## # i 9 variables: tailnum <chr>, year <int>, type <chr>, mar
## #   model <chr>, engines <int>, seats <int>, speed <int>, e
```

Key variables

- Check if there are any **missing values** in the primary key.If there are, then it can't identify an observation either!

```
planes |>
  filter(is.na(tailnum))
```

```
## # A tibble: 0 x 9
## # i 9 variables: tailnum <chr>, year <int>, type <chr>, mar
## #   model <chr>, engines <int>, seats <int>, speed <int>, e
```

```
weather |>
  filter(is.na(time_hour) | is.na(origin))
```

```
## # A tibble: 0 x 15
## # i 15 variables: origin <chr>, year <int>, month <int>, da
## #   temp <dbl>, dewp <dbl>, humid <dbl>, wind_dir <dbl>, wi
## #   wind_gust <dbl>, precip <dbl>, pressure <dbl>, visib <d
## #   time_hour <dttm>
```

Key variables

- Sometimes you can't use the primary key to identify observations, but you can use a combination of variables.

```
weather %>%
  count(year,month,day,hour,origin) %>%
  filter(n>1)
```

```
## # A tibble: 3 x 6
##    year month   day  hour origin     n
##   <int> <int> <int> <int> <chr>  <int>
## 1  2013    11     3     1 EWR        2
## 2  2013    11     3     1 JFK        2
## 3  2013    11     3     1 LGA        2
```

Join(merge) tables

- A primary key and the corresponding foreign key in another table form a **relation**.
  - **1 : M(any)**
  - **1 : 1**
  - **Many : Many**
- 1:1 and 1:M are the most common relations in practice.
- Many:Many relations are rare.

Join(merge) tables

- **mutating join**: It first matches observations by their keys, then copies across variables from one table to the other.
- Aim: add the full `airline` name to the `flight2`

```
flights2 <- flights %>%
  select(year:day,hour,tailnum,carrier,origin)
```

Join(merge) tables

- Aim: add the full `airline` name to the `flight2`

```
flights2  %>%
  left_join(airlines, by = "carrier")
```

```
## # A tibble: 336,776 x 8
##     year month   day  hour tailnum carrier origin name
##    <int> <int> <int> <dbl> <chr>   <chr>   <chr>  <chr>
##  1  2013     1     1     5 N14228  UA      EWR    United Air Lines Inc.
##  2  2013     1     1     5 N24211  UA      LGA    United Air Lines Inc.
##  3  2013     1     1     5 N619AA  AA      JFK    American Airlines Inc.
##  4  2013     1     1     5 N804JB  B6      JFK    JetBlue Airways
##  5  2013     1     1     6 N668DN  DL      LGA    Delta Air Lines Inc.
##  6  2013     1     1     5 N39463  UA      EWR    United Air Lines Inc.
##  7  2013     1     1     6 N516JB  B6      EWR    JetBlue Airways
##  8  2013     1     1     6 N829AS  EV      LGA    ExpressJet Airlines In
##  9  2013     1     1     6 N593JB  B6      JFK    JetBlue Airways
## 10  2013     1     1     6 N3ALAA  AA      LGA    American Airlines Inc.
## # i 336,766 more rows
```

Join(merge) tables

```
planes %>%
  select(tailnum,type) %>%
  right_join(flights,by = "tailnum")
```

```
## # A tibble: 336,776 x 20
##    tailnum type      year month   day dep_time sched_dep_time dep_delay
##    <chr>   <chr>    <int> <int> <int>    <int>          <int>     <dbl>
##  1 N10156  Fixed w~  2013     1    10      626            630        -4
##  2 N10156  Fixed w~  2013     1    10     1120           1032        48
##  3 N10156  Fixed w~  2013     1    10     1619           1540        39
##  4 N10156  Fixed w~  2013     1    11      632            634        -2
##  5 N10156  Fixed w~  2013     1    11     1116           1120        -4
##  6 N10156  Fixed w~  2013     1    11     1845           1819        26
##  7 N10156  Fixed w~  2013     1    12      830            838        -8
##  8 N10156  Fixed w~  2013     1    12     1410           1359        11
##  9 N10156  Fixed w~  2013     1    13     1551           1536        15
## 10 N10156  Fixed w~  2013     1    13     2221           2039       102
## # i 336,766 more rows
## # i 11 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <c
## #   flight <int>, origin <chr>, dest <chr>, air_time <dbl>, distance <db
```

Practice 1: combine flights and weather

- combine flights and weather to find the weather conditions for each flight.

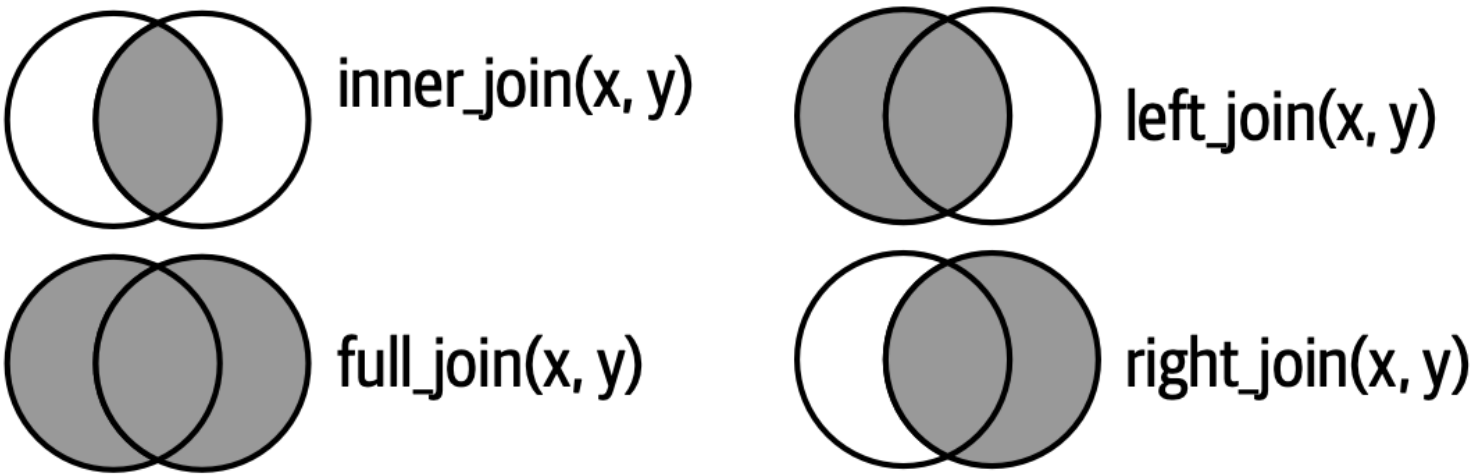
```
flights2 %>%
  left_join(weather, by = c("year", "month", "day", "hour", "origin"))
```

```
## # A tibble: 336,776 x 17
##     year month   day  hour tailnum carrier origin  temp  dewp humid wind
##    <int> <int> <int> <dbl> <chr>   <chr>   <chr>  <dbl> <dbl> <dbl>    <
##  1  2013     1     1     5 N14228  UA      EWR     39.0  28.0  64.4
##  2  2013     1     1     5 N24211  UA      LGA     39.9  25.0  54.8
##  3  2013     1     1     5 N619AA  AA      JFK     39.0  27.0  61.6
##  4  2013     1     1     5 N804JB  B6      JFK     39.0  27.0  61.6
##  5  2013     1     1     6 N668DN  DL      LGA     39.9  25.0  54.8
##  6  2013     1     1     5 N39463  UA      EWR     39.0  28.0  64.4
##  7  2013     1     1     6 N516JB  B6      EWR     37.9  28.0  67.2
##  8  2013     1     1     6 N829AS  EV      LGA     39.9  25.0  54.8
##  9  2013     1     1     6 N593JB  B6      JFK     37.9  27.0  64.3
## 10  2013     1     1     6 N3ALAA  AA      LGA     39.9  25.0  54.8
## # i 336,766 more rows
```

Further understanding joins

```
x <- tribble(
  ~key, ~val_x,
     1, "x1",
     2, "x2",
     3, "x3"
)
y <- tribble(
  ~key, ~val_y,
     1, "y1",
     2, "y2",
     4, "y3"
)
```

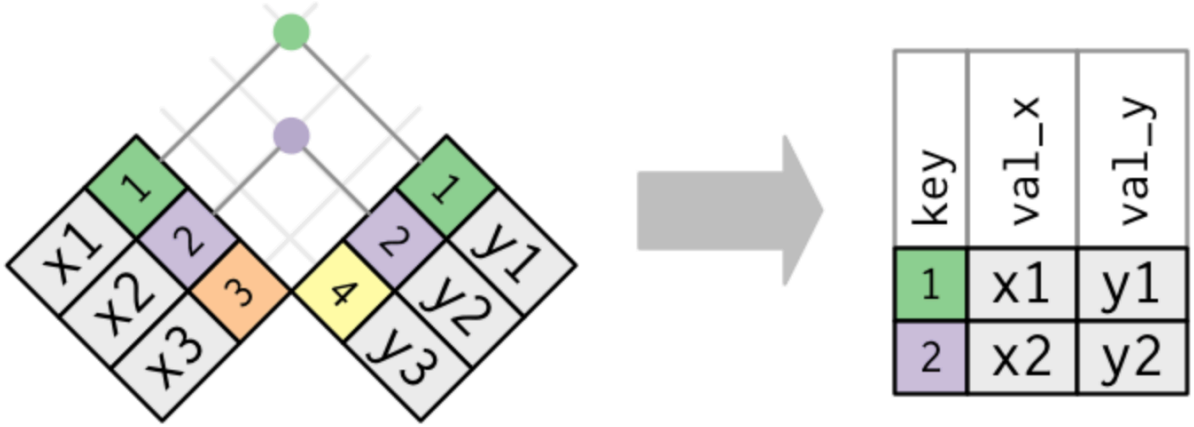
Many joins



- Inner_join
- Left_join
- Right_join
- Full_join

Inner joins

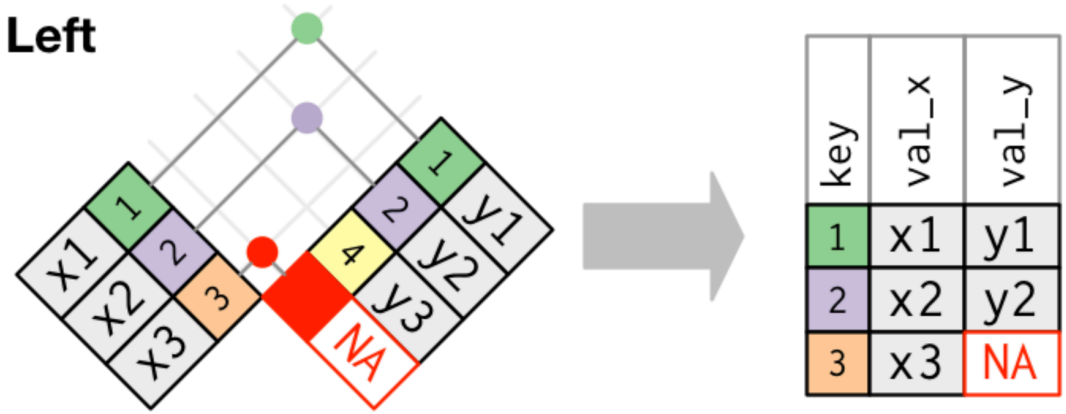
- **Inner join** matches pairs of observations whenever their keys are **equal**.




```
## # A tibble: 2 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
```

Outer joins:left

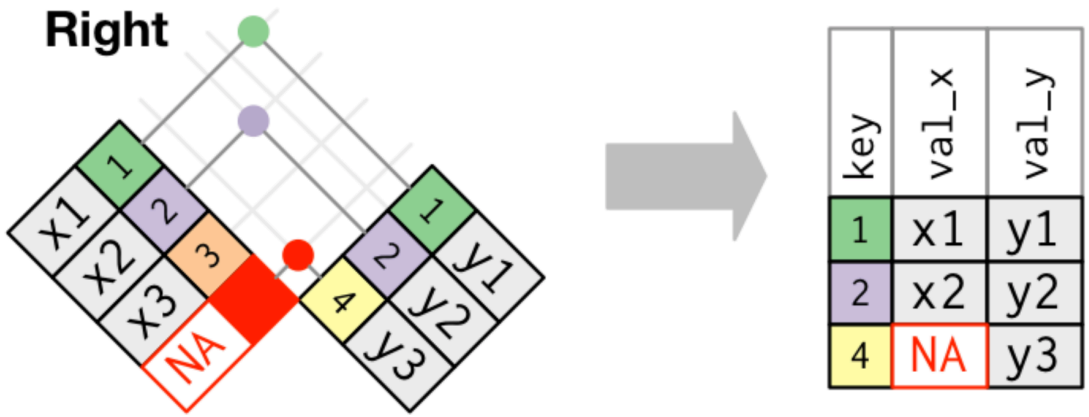
- A **left join** keeps all observations in x.




```
## # A tibble: 2 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
```

Outer joins: right

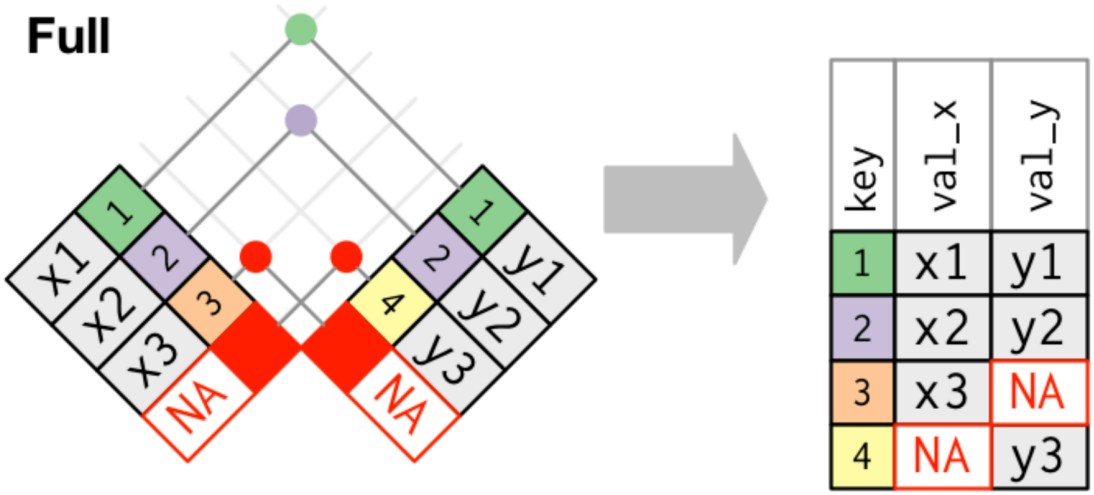
- A **right join** keeps all observations in y.




```
## # A tibble: 2 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
```

Outer joins: full

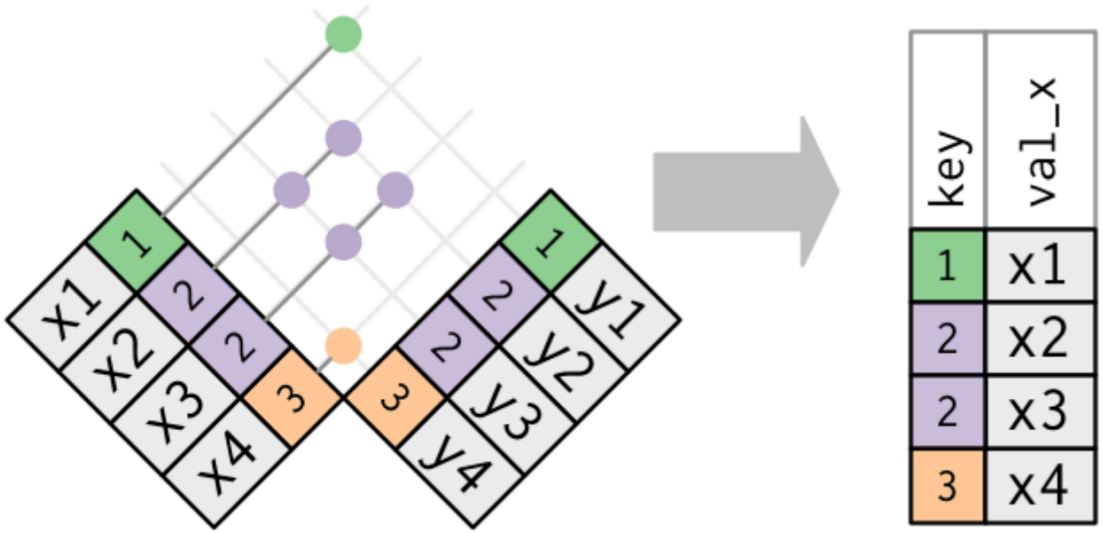
- A **full join** keeps all observations in x and y.




```
## # A tibble: 4 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     3 x3    <NA>
## 4     4 <NA>  y3
```

Filtering joins: `semi_join`

- Only the existence of a **match** is important, while it does not matter which observation is matched.
- This means that filtering joins never duplicate rows like mutating joins do:




```
## # A tibble: 2 x 2
##     key val_x
##   <dbl> <chr>
```

Filtering joins: `anti_join`

- `anti_join(x,y)` **drops** all observations in x that have a match in y.


```
## # A tibble: 1 x 2
##     key val_x
##   <dbl> <chr>
## 1     3 x3
```

More joins: Non-equi joins

- Cross join
- Inequality joins
- Rolling joins
- Overlap joins

Top 10 most popular destinations

```
## # A tibble: 10 x 2
##    dest      n
##    <chr> <int>
##  1 ORD   17283
##  2 ATL   17215
##  3 LAX   16174
##  4 BOS   15508
##  5 MCO   14082
##  6 CLT   14064
##  7 SFO   13331
##  8 FLL   12055
##  9 MIA   11728
## 10 DCA    9705
```

Top 10 most popular destinations

- find each flight that went to one of those destinations.

```
## # A tibble: 141,145 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_ar
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      542            540         2      923
##  2  2013     1     1      554            600        -6      812
##  3  2013     1     1      554            558        -4      740
##  4  2013     1     1      555            600        -5      913
##  5  2013     1     1      557            600        -3      838
##  6  2013     1     1      558            600        -2      753
##  7  2013     1     1      558            600        -2      924
##  8  2013     1     1      558            600        -2      923
##  9  2013     1     1      559            559         0      702
## 10  2013     1     1      600            600         0      851
## # i 141,135 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <d
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Some Notes for join data

1. Start by identifying the variables that form the primary key in each table.
2. Check that none of the variables in the primary key are missing and whether is duplicated(for inner join).
3. Check that your foreign keys match primary keys in another table.

Section 2

Introduction to Reshape data

Data Structure

- **Tidy data**
- **Long data**
- **Wide data**

How to transform from wide to long data

```
df <- tribble(
  ~id,  ~bp1, ~bp2,
   "A",  100,  120,
   "B",  140,  115,
   "C",  120,  125
)
```

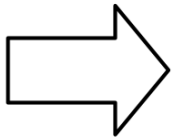
How to transform from wide to long data

```
df %>%
  pivot_longer(
    cols = bp1:bp2,
    names_to = "measurement",
    values_to = "value"
  )
```

```
## # A tibble: 6 x 3
##   id    measurement value
##   <chr> <chr>       <dbl>
## 1 A     bp1           100
## 2 A     bp2           120
## 3 B     bp1           140
## 4 B     bp2           115
## 5 C     bp1           120
## 6 C     bp2           125
```

What happens when we transform from long to wide data

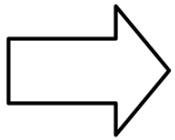
id	bp1	bp2
A	100	120
B	140	115
C	120	125



id	measurement	value
A	bp1	100
A	bp2	120
B	bp1	140
B	bp2	115
C	bp1	120
C	bp2	125

What happens when we transform from long to wide data

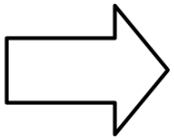
id	bp1	bp2
A	100	120
B	140	115
C	120	125



id	measurement	value
A	bp1	100
A	bp2	120
B	bp1	140
B	bp2	115
C	bp1	120
C	bp2	125

What happens when we transform from long to wide data

id	bp1	bp2
A	100	120
B	140	115
C	120	125



id	measurement	value
A	bp1	100
A	bp2	120
B	bp1	140
B	bp2	115
C	bp1	120
C	bp2	125

gapminder data

```
df<- readr::read_csv("https://byelenin.github.io/QSS_2025/data
```

- from wide to long

```
gapminder_long <- df %>%
  pivot_longer(cols = country, names_to = "year", values_to =
```

- from long to wide

```
gapminder_wide2<- gapminder_long %>%
  pivot_wider(names_from = year, values_from = value)
```

Reference

1. Grolemund & Wickham(2023), R for Data Science(2nd edition)
2. DataCamp(2018), Introduction to Tidyverse