10 - Relational Data and Joins

ST 597 | Spring 2017
University of Alabama

10-relational.pdf

Contents

Required Packages and Data

```
library(tidyverse)
library(nycflights13)
library(Lahman)
library(babynames)
library(fueleconomy)
library(nasaweather)
```

1 Relational Data

We are going to follow the discussion in Chapter 13 Relational Data from the R for Data Science book.

1.1 nycflights13

Load the `nycflights13` package and check out the available datasets.

```
library(nycflights13)              # load package
data(package='nycflights13')       # shows datasets

# airlines                        Airline names.
# airports                        Airport metadata
# flights                         Flights data
# planes                          Plane metadata.
# weather                         Hourly weather data
```

Print out the column names as a list

```
list(airlines = colnames(airlines),
     airports = colnames(airports),
     flights = colnames(flights),
     planes = colnames(planes),
     weather = colnames(weather))
#> $airlines
#> [1] "carrier" "name"
#>
#> $airports
#> [1] "faa"  "name" "lat"  "lon"  "alt"  "tz"   "dst"
#>
#> $flights
#>  [1] "year"           "month"          "day"            "dep_time"
#>  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
#>  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
#> [13] "origin"         "dest"           "air_time"       "distance"
#> [17] "hour"           "minute"         "time_hour"
#>
#> $planes
#> [1] "tailnum"      "year"         "type"         "manufacturer"
#> [5] "model"        "engines"      "seats"        "speed"
#> [9] "engine"
#>
#> $weather
#>  [1] "origin"     "year"       "month"      "day"        "hour"
#>  [6] "temp"       "dewp"       "humid"      "wind_dir"   "wind_speed"
#> [11] "wind_gust"  "precip"     "pressure"   "visib"      "time_hour"
```

https://raw.githubusercontent.com/hadley/r4ds/master/diagrams/relational-nycflights.png

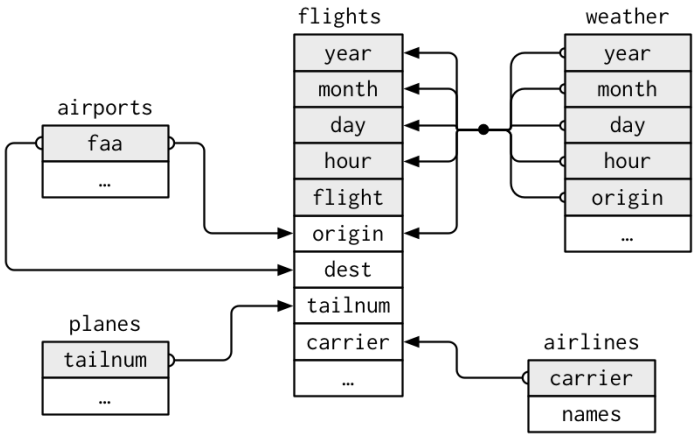

Figure 1: nycflights13 relations

1.2 Exercises

1. Imagine you want to draw (approximately) the route each plane flies from its origin to its destination. What variables would you need? What tables would you need to combine?
2. I forgot to draw the a relationship between `weather` and `airports`.
   What is the relationship and what should it look like in the diagram?
3. `weather` only contains information for the origin (NYC) airports. If it contained weather records for all airports in the USA, what additional relation would it define with `flights`?

1.2.1 Your Turn: Relations

Your Turn #1 : Relations

1. You might expect that there is an implicit relationship between `planes` and `airlines`, because each plane is flown by a single airline. Confirm or reject this hypothesis using data.
   - Can `planes` and `airlines` be directly connected?
   - How could `planes` and `airlines` be connected from the `flights` data?
   - Do some planes (`tailnum`) have multiple carriers? How can we find out with the `flights` data?
2. We know that some days of the year are "special", and fewer people than usual fly on them.
   - Represent this data as a data frame?
   - What would be the primary keys of that table? How would it connect to the existing tables?

1.3 Keys (R4DS 13.3)

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.

There are two types of keys:

- A **primary key** uniquely identifies an observation in its own table. For example, `planes$tailnum` is a primary key because it uniquely identifies each plane in the `planes` table.
- A **foreign key** uniquely identifies an observation in another table. For example, the `flights$tailnum` is a foreign key because it appears in the `flights` table where it matches each flight to a unique plane.

We can check for (verify) a primary key with the code

```
count(<data>, <keys>) %>% filter(n>1)
```

1.3.1 Exercises

1. What is the primary key for flights dataset?
2. Add a surrogate key to flights.
3. Identify the keys in the `Lahman::Batting` dataset. Hint, convert `Batting` to tibble to help with printing.
4. Draw a diagram illustrating the connections between the `Batting`, `Master`, and `Salaries` tables in the Lahman package.
5. How would you characterise the relationship between the `Batting`, `Pitching`, and `Fielding` tables?

1.3.2 Your Turn: Keys

Your Turn #2 : Keys

Identify the keys in the following datasets:

1. `babynames::babynames`
2. `nasaweather::atmos`
3. `fueleconomy::vehicles`

2 Joins

Joins are used to combine or merge two datasets. This is a major aspect of SQL. While the base function `merge()` can also do some of these things, we will examine the functions available from the `dplyr` package.

The Data Transformation Cheatsheet is a good reference.

There are two main types of joins: **mutating** joins add columns and **filtering** joins remove rows. It not this simple, but this will get you started.

2.1 Mutating joins (R4DS 13.4)

Make the `flights2` data.

```
(flights2 <- flights %>% select(year:day, hour, origin, dest, tailnum, carrier))
#> # A tibble: 336,776 × 8
#>     year month   day  hour origin  dest tailnum carrier
#>    <int> <int> <int> <dbl>  <chr> <chr>   <chr>   <chr>
#> 1   2013     1     1     5    EWR   IAH  N14228      UA
#> 2   2013     1     1     5    LGA   IAH  N24211      UA
#> 3   2013     1     1     5    JFK   MIA  N619AA      AA
#> 4   2013     1     1     5    JFK   BQN  N804JB      B6
#> 5   2013     1     1     6    LGA   ATL  N668DN      DL
#> 6   2013     1     1     5    EWR   ORD  N39463      UA
#> 7   2013     1     1     6    EWR   FLL  N516JB      B6
#> 8   2013     1     1     6    LGA   IAD  N829AS      EV
#> 9   2013     1     1     6    JFK   MCO  N593JB      B6
#> 10  2013     1     1     6    LGA   ORD  N3ALAA      AA
#> # ... with 336,766 more rows
```

Join `flights2` with the `airlines` data.

```
#- Solution using joins
flights2 %>%
  select(-origin, -dest) %>%
  left_join(airlines, by = "carrier")
#> # A tibble: 336,776 × 7
#>     year month   day  hour tailnum carrier                    name
#>    <int> <int> <int> <dbl>   <chr>   <chr>                   <chr>
#> 1   2013     1     1     5  N14228      UA    United Air Lines Inc.
#> 2   2013     1     1     5  N24211      UA    United Air Lines Inc.
#> 3   2013     1     1     5  N619AA      AA   American Airlines Inc.
#> 4   2013     1     1     5  N804JB      B6          JetBlue Airways
#> 5   2013     1     1     6  N668DN      DL     Delta Air Lines Inc.
#> 6   2013     1     1     5  N39463      UA    United Air Lines Inc.
#> 7   2013     1     1     6  N516JB      B6          JetBlue Airways
#> 8   2013     1     1     6  N829AS      EV ExpressJet Airlines Inc.
#> 9   2013     1     1     6  N593JB      B6          JetBlue Airways
#> 10  2013     1     1     6  N3ALAA      AA   American Airlines Inc.
#> # ... with 336,766 more rows
```

Alternative solutions

```
#- explicit argument names
left_join(x = flights2, y = airlines, by = "carrier")

#- Solution using match() and indexing
flights2 %>%
  mutate(name = airlines$name[match(carrier, airlines$carrier)])
```

Mutating Joins See 13.4 of R4DS

- `inner_join(x, y)` only includes observations that having matching x and y key values. Rows of x can be dropped/filtered.
- `left_join(x, y)` includes all observations in x, regardless of whether they match or not. This is the most commonly used join because it ensures that you don't lose observations from your primary table.
- `right_join(x, y)` includes all observations in y. It's equivalent to `left_join(y, x)`, but the columns will be ordered differently.
- `full_join()` includes all observations from x and y.
- The left, right and full joins are collectively know as **outer joins**. When a row doesn't match in an outer join, the new variables are filled in with missing values.
  - **outer joins** will fill any missing values with `NA`
- If there are duplicate keys, all combinations are returned.
- Missing values are given `NA`.

2.1.1 Defining the Key Columns (R4DS 13.4.5)

Check out the help for a join to see its arguments.

```
?inner_join
```

Notice that the `by=` argument is set to `NULL` which indicates a **natural join**. A natural join uses all variables with common names across the two tables.

For example,

```
left_join(x=flights2, y=weather)       # flights2 %>% left_join(weather)
#> Joining, by = c("year", "month", "day", "hour", "origin")
#> # A tibble: 336,776 × 18
#>     year month   day  hour origin  dest tailnum carrier  temp  dewp humid
#>    <dbl> <dbl> <int> <dbl>  <chr> <chr>   <chr>   <chr> <dbl> <dbl> <dbl>
#> 1   2013     1     1     5    EWR   IAH  N14228      UA    NA    NA    NA
#> 2   2013     1     1     5    LGA   IAH  N24211      UA    NA    NA    NA
#> 3   2013     1     1     5    JFK   MIA  N619AA      AA    NA    NA    NA
#> 4   2013     1     1     5    JFK   BQN  N804JB      B6    NA    NA    NA
#> 5   2013     1     1     6    LGA   ATL  N668DN      DL 39.92 26.06 57.33
#> 6   2013     1     1     5    EWR   ORD  N39463      UA    NA    NA    NA
#> 7   2013     1     1     6    EWR   FLL  N516JB      B6 39.02 26.06 59.37
#> 8   2013     1     1     6    LGA   IAD  N829AS      EV 39.92 26.06 57.33
#> 9   2013     1     1     6    JFK   MCO  N593JB      B6 39.02 26.06 59.37
#> 10  2013     1     1     6    LGA   ORD  N3ALAA      AA 39.92 26.06 57.33
#> # ... with 336,766 more rows, and 7 more variables: wind_dir <dbl>,
#> #   wind_speed <dbl>, wind_gust <dbl>, precip <dbl>, pressure <dbl>,
#> #   visib <dbl>, time_hour <dttm>
```

And notice the message `Joining by: c("year", "month", "day", "hour", "origin")`, which indicates the variables used for joining. This is equivalent to explicitly using

```
left_join(flights2, weather, by = c("year", "month", "day", "hour", "origin"))
#> # A tibble: 336,776 × 18
```

```
#>     year month   day  hour origin  dest tailnum carrier  temp  dewp humid
#>    <dbl> <dbl> <int> <dbl>  <chr> <chr>   <chr>   <chr> <dbl> <dbl> <dbl>
#> 1   2013     1     1     5    EWR   IAH  N14228      UA    NA    NA    NA
#> 2   2013     1     1     5    LGA   IAH  N24211      UA    NA    NA    NA
#> 3   2013     1     1     5    JFK   MIA  N619AA      AA    NA    NA    NA
#> 4   2013     1     1     5    JFK   BQN  N804JB      B6    NA    NA    NA
#> 5   2013     1     1     6    LGA   ATL  N668DN      DL 39.92 26.06 57.33
#> 6   2013     1     1     5    EWR   ORD  N39463      UA    NA    NA    NA
#> 7   2013     1     1     6    EWR   FLL  N516JB      B6 39.02 26.06 59.37
#> 8   2013     1     1     6    LGA   IAD  N829AS      EV 39.92 26.06 57.33
#> 9   2013     1     1     6    JFK   MCO  N593JB      B6 39.02 26.06 59.37
#> 10  2013     1     1     6    LGA   ORD  N3ALAA      AA 39.92 26.06 57.33
#> # ... with 336,766 more rows, and 7 more variables: wind_dir <dbl>,
#> #   wind_speed <dbl>, wind_gust <dbl>, precip <dbl>, pressure <dbl>,
#> #   visib <dbl>, time_hour <dttm>
```

It is always to good to set `by=`, so you don't get any unintentional results, like this

```
left_join(flights2, planes, by = NULL)
#> Joining, by = c("year", "tailnum")
#> # A tibble: 336,776 × 15
#>     year month   day  hour origin  dest tailnum carrier  type manufacturer
#>    <int> <int> <int> <dbl>  <chr> <chr>   <chr>   <chr> <chr>        <chr>
#> 1   2013     1     1     5    EWR   IAH  N14228      UA  <NA>         <NA>
#> 2   2013     1     1     5    LGA   IAH  N24211      UA  <NA>         <NA>
#> 3   2013     1     1     5    JFK   MIA  N619AA      AA  <NA>         <NA>
#> 4   2013     1     1     5    JFK   BQN  N804JB      B6  <NA>         <NA>
#> 5   2013     1     1     6    LGA   ATL  N668DN      DL  <NA>         <NA>
#> 6   2013     1     1     5    EWR   ORD  N39463      UA  <NA>         <NA>
#> 7   2013     1     1     6    EWR   FLL  N516JB      B6  <NA>         <NA>
#> 8   2013     1     1     6    LGA   IAD  N829AS      EV  <NA>         <NA>
#> 9   2013     1     1     6    JFK   MCO  N593JB      B6  <NA>         <NA>
#> 10  2013     1     1     6    LGA   ORD  N3ALAA      AA  <NA>         <NA>
#> # ... with 336,766 more rows, and 5 more variables: model <chr>,
#> #   engines <int>, seats <int>, speed <int>, engine <chr>
```

Why all the `NA`'s?

Notice that `flights` has a `year` column that refers to the year of the flight. The `planes` also has a `year` column, but this refers to the year manufactured. Not many flights with a plane that is just made. What we really want is to joining `by = 'tailnum'` only:

```
left_join(flights2, planes, by = "tailnum")
#> # A tibble: 336,776 × 16
#>    year.x month   day  hour origin  dest tailnum carrier year.y
#>     <int> <int> <int> <dbl>  <chr> <chr>   <chr>   <chr>  <int>
#> 1    2013     1     1     5    EWR   IAH  N14228      UA   1999
#> 2    2013     1     1     5    LGA   IAH  N24211      UA   1998
#> 3    2013     1     1     5    JFK   MIA  N619AA      AA   1990
#> 4    2013     1     1     5    JFK   BQN  N804JB      B6   2012
#> 5    2013     1     1     6    LGA   ATL  N668DN      DL   1991
#> 6    2013     1     1     5    EWR   ORD  N39463      UA   2012
#> 7    2013     1     1     6    EWR   FLL  N516JB      B6   2000
#> 8    2013     1     1     6    LGA   IAD  N829AS      EV   1998
```

```
#> 9    2013     1     1     6    JFK   MCO  N593JB      B6   2004
#> 10   2013     1     1     6    LGA   ORD  N3ALAA      AA     NA
#> # ... with 336,766 more rows, and 7 more variables: type <chr>,
#> #   manufacturer <chr>, model <chr>, engines <int>, seats <int>,
#> #   speed <int>, engine <chr>
```

And notice that because of the conflict, the `year` variable is no longer. Instead, the `year.x` variables is the `year` from the `flights2` data and the `year.y` variable represents the `year` from the `planes` data.

If the same key has different names between the two tables, then a *named character vector* can be used. Recall the `airports` data has a key column `faa` that indicates the FAA airport code. This links to the `origin` and `dest` fields in the `flights2` data.

```
#- join airports$faa to flights2$dest
left_join(flights2, airports, c("dest" = "faa"))
#> # A tibble: 336,776 × 14
#>     year month   day  hour origin  dest tailnum carrier
#>    <int> <int> <int> <dbl>  <chr> <chr>   <chr>   <chr>
#> 1   2013     1     1     5    EWR   IAH  N14228      UA
#> 2   2013     1     1     5    LGA   IAH  N24211      UA
#> 3   2013     1     1     5    JFK   MIA  N619AA      AA
#> 4   2013     1     1     5    JFK   BQN  N804JB      B6
#> 5   2013     1     1     6    LGA   ATL  N668DN      DL
#> 6   2013     1     1     5    EWR   ORD  N39463      UA
#> 7   2013     1     1     6    EWR   FLL  N516JB      B6
#> 8   2013     1     1     6    LGA   IAD  N829AS      EV
#> 9   2013     1     1     6    JFK   MCO  N593JB      B6
#> 10  2013     1     1     6    LGA   ORD  N3ALAA      AA
#> # ... with 336,766 more rows, and 6 more variables: name <chr>, lat <dbl>,
#> #   lon <dbl>, alt <int>, tz <dbl>, dst <chr>
```

Do you know why there are `NA`'s? What if we used `inner_join()` instead of `left_join()`? What would happen to the NA's?

```
inner_join(flights2, airports, c("dest" = "faa"))
```

Here we join to the `origin` instead of `dest`

```
#- join airports$faa to flights2$origin
left_join(flights2, airports, c("origin" = "faa"))
#> # A tibble: 336,776 × 14
#>     year month   day  hour origin  dest tailnum carrier
#>    <int> <int> <int> <dbl>  <chr> <chr>   <chr>   <chr>
#> 1   2013     1     1     5    EWR   IAH  N14228      UA
#> 2   2013     1     1     5    LGA   IAH  N24211      UA
#> 3   2013     1     1     5    JFK   MIA  N619AA      AA
#> 4   2013     1     1     5    JFK   BQN  N804JB      B6
#> 5   2013     1     1     6    LGA   ATL  N668DN      DL
#> 6   2013     1     1     5    EWR   ORD  N39463      UA
#> 7   2013     1     1     6    EWR   FLL  N516JB      B6
#> 8   2013     1     1     6    LGA   IAD  N829AS      EV
#> 9   2013     1     1     6    JFK   MCO  N593JB      B6
#> 10  2013     1     1     6    LGA   ORD  N3ALAA      AA
```

```
#> # ... with 336,766 more rows, and 6 more variables: name <chr>, lat <dbl>,
#> #   lon <dbl>, alt <int>, tz <dbl>, dst <chr>
```

2.1.2 Exercises

1. Compute the average delay by destination, then join on the `airports` data frame so you can show the spatial distribution of delays. (We will learn how to draw such maps later in the course).

2.2 Filtering Joins (R4DS 13.5)

Filtering joins match observations in the same way as mutating joins, but affect the observations, not the variables. There are two types:

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

A semi-join connects two tables like a mutating join, but instead of adding new columns, only keeps the rows in `x` that have a match in `y`. An anti-join is the reverse, it keeps the rows in `x` that do **not** have a match in `y`.

2.2.1 Your Turn: Joins

Your Turn #3 : Joins

1. Is there a relationship between the age of a plane and its *average* delays?
2. What does `anti_join(flights, airports, by = c("dest" = "faa"))` tell you? What does `anti_join(airports, flights, by = c("faa" = "dest"))` tell you?
3. Filter `flights` to only show flights with planes that have flown at least 100 flights.
4. Find all the planes (`tailnum`) manufacturered by `AIRBUS` and flown by `Delta`.

3 Join Problems (R4DS 13.6)

4 Set Operations (R4DS 13.7)

The final type of two-table verb is set operations. These expect the x and y inputs to have the same columns, and treats the observations like sets:

- `intersect(x, y)`: return only observations in both x and y
- `union(x, y)`: return unique observations in x and y
- `setdiff(x, y)`: return observations in x, but not in y.

5 SQL Correspondence

SQL is the inspiration for dplyr's conventions, so the translation is straightforward:

Each two-table verb has a straightforward SQL equivalent:

dplyr	SQL
`inner_join(x, y, by = "z")`	`SELECT * FROM x INNER JOIN y USING (z)`
`left_join(x, y, by = "z")`	`SELECT * FROM x LEFT OUTER JOIN y USING (z)`
`right_join(x, y, by = "z")`	`SELECT * FROM x RIGHT OUTER JOIN y USING (z)`
`full_join(x, y, by = "z")`	`SELECT * FROM x FULL OUTER JOIN y USING (z)`
`semi_join()`	`SELECT * FROM x WHERE EXISTS (SELECT 1 FROM y WHERE x.a = y.a)`

dplyr	SQL
`anti_join()`	`SELECT * FROM x WHERE NOT EXISTS (SELECT 1 FROM y WHERE x.a = y.a)`
`intersect(x, y)`	`SELECT * FROM x INTERSECT SELECT * FROM y`
`union(x, y)`	`SELECT * FROM x UNION SELECT * FROM y`
`setdiff(x, y)`	`SELECT * FROM x EXCEPT SELECT * FROM y`

Note that “INNER” and “OUTER” are optional, and often omitted.