# 12 - Relational Data and Joins

Data and Information Engineering

*SYS 2202 | Fall 2019*

*12-relational.pdf*

## Contents

---

Some figures and text from this chapter are taken from R for Data Science by Garrett Grolemund & Hadley Wickham

#### Required Packages and Data

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

# 1 Relational Data

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

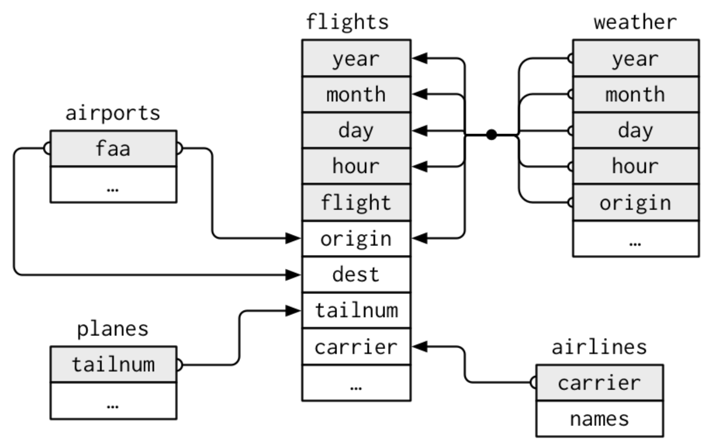
## 1.1 nycflights13

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"   "tzone"
#>
#> $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://github.com/hadley/r4ds/blob/master/diagrams/relational-nycflights.png

## 1.2 Exercises

1. Imagine you want to draw a line for 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?

## 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 also a foreign key because it appears in the `flights` table where it matches each flight to a unique plane.

### 1.3.1 Primary Keys

- A primary key can made from multiple columns. This is called a *composite primary key*.
  - For example, the `weather` table (should) have a primary key of: `origin`, `year`, `month`, `day`, `hour` (but see below to see if it really does)
- The *primary key* column(s) must have unique values; there shouldn't be any duplicates.
  - There also can't be any missing (`NA`) or `NULL` values
- If there is not a *natural* primary key, then we can create a *surrogate key*. This is simply a unique identifier for each row.

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

```
count(<data>, <keys>) %>% filter(n>1) # this should be empty if primary key
```

For example,

```
planes %>% count(tailnum) %>% filter(n>1)
#> # A tibble: 0 x 2
#> # ... with 2 variables: tailnum <chr>, n <int>
weather %>% count(origin, year, month, day, hour) %>% filter(n>1)
#> # A tibble: 3 x 6
#>   origin  year month   day  hour     n
#>   <chr>  <dbl> <dbl> <int> <int> <int>
#> 1 EWR     2013    11     3     1     2
#> 2 JFK     2013    11     3     1     2
#> 3 LGA     2013    11     3     1     2
# Note: not unique! there were multiple measures at the same time.
```

**Column Summaries**

If we want to check if any *single column* could be a primary key (e.g., has unique values), we can use the `summarize_all()` function.

```
#-- Find number of unique values in all columns
airports %>% summarize_all(n_distinct)
#> # A tibble: 1 x 8
#>     faa  name   lat   lon   alt    tz   dst tzone
#>   <int> <int> <int> <int> <int> <int> <int> <int>
#> 1  1458  1440  1456  1458   911     7     3    10

#-- Find if any columns have all unique values
airports %>%
  summarize_all(function(x) n_distinct(x) == length(x)) %>%
  gather(column, key)   # convert to long format
#> # A tibble: 8 x 2
#>   column key
#>   <chr>  <lgl>
#> 1 faa    TRUE
#> 2 name   FALSE
#> 3 lat    FALSE
#> 4 lon    TRUE
#> 5 alt    FALSE
#> 6 tz     FALSE
#> # ... with 2 more rows
```

There are also `summarize_if()`, and `summarize_at()` functions that can simplify code when you want to apply the same function(s) to many columns:

```r
#-- Get the mean value for all *numeric* columns
flights %>%
  summarize_if(is.numeric, mean, na.rm=TRUE) %>%
  gather(column, mean)  # convert to long format
#> # A tibble: 14 x 2
#>   column            mean
#>   <chr>            <dbl>
#> 1 year            2013
#> 2 month              6.55
#> 3 day               15.7
#> 4 dep_time        1349.
#> 5 sched_dep_time 1344.
#> 6 dep_delay         12.6
#> # ... with 8 more rows
```

### 1.3.2 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?

### 1.3.3 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.
- The chapter R4DS: Relational Data has helpful details.

There are two main types of joins:

1. **mutating** joins add columns
2. **filtering** joins remove rows

## 2.1 Mutating joins (R4DS 13.4)

The idea of a *mutating join* is to combine information (i.e., columns) from two tables.

- To do this, the function will need to know how the *rows* are connected. E.g., row 3 from Table 1 is connected to row 10 from Table 2.
  - Thus, joins will use *primary* and *foreign* keys to connect the rows

Make the `flights2` data (fewer columns so we can better see the new columns)

```
(flights2 <- flights %>% select(year:day, hour, origin, dest, tailnum, carrier))
#> # A tibble: 336,776 x 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
#> # ... with 3.368e+05 more rows
```

Join `flights2` with the `airlines` data to add the airline `name`

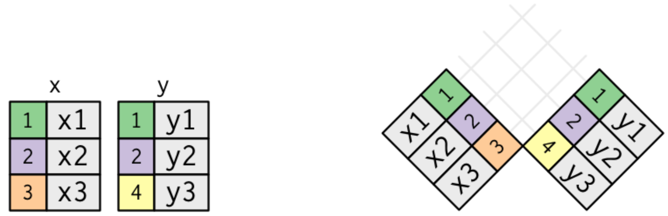
```
#- Solution using joins
flights2 %>%
  left_join(airlines, by = "carrier") # use the `carrier` column
#> # A tibble: 336,776 x 9
#>    year month   day  hour origin dest  tailnum carrier name
#>   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr>
#> 1  2013     1     1     5 EWR    IAH   N14228  UA      United Air Lines In~
#> 2  2013     1     1     5 LGA    IAH   N24211  UA      United Air Lines In~
#> 3  2013     1     1     5 JFK    MIA   N619AA  AA      American Airlines I~
#> 4  2013     1     1     5 JFK    BQN   N804JB  B6      JetBlue Airways
#> 5  2013     1     1     6 LGA    ATL   N668DN  DL      Delta Air Lines Inc.
#> 6  2013     1     1     5 EWR    ORD   N39463  UA      United Air Lines In~
#> # ... with 3.368e+05 more rows
```

Alternative solutions

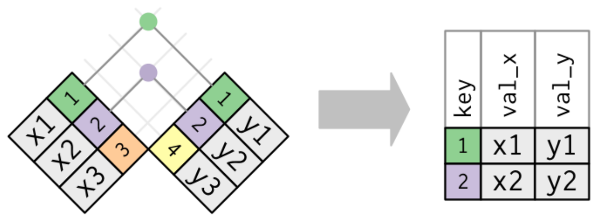
```
#- explicit argument names
left_join(x = flights2, y = airlines, by = "carrier")

#- Solution using match() and indexing
```

```r
flights2 %>%
  mutate(name = airlines$name[match(carrier, airlines$carrier)])
```

**Mutating Joins** See 13.4 of R4DS

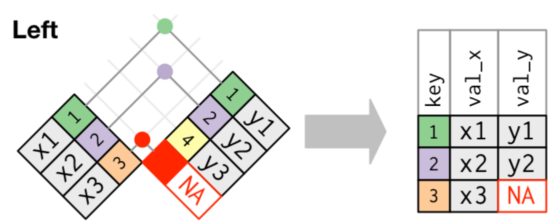



- `inner_join(x, y)` only includes observations having *matching x and y key values*.
  - Note: Rows of x can be dropped/filtered.

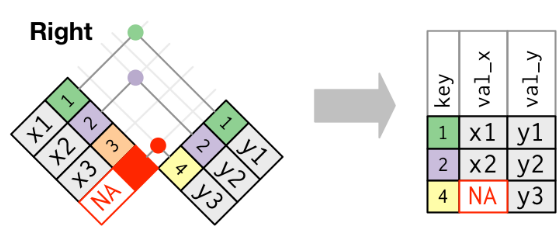



- The `left_join()`, `right_join()` and `full_join()` are collectively know as **outer joins**.
  - When a row doesn't match in an outer join, the new variables are filled in with missing values.
  - **outer joins** will fill any missing values with `NA`

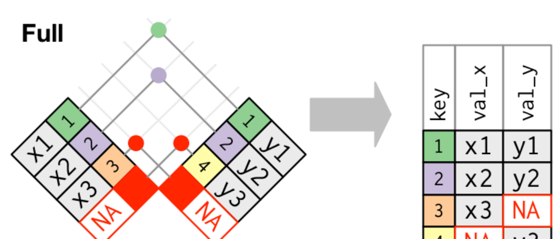
- `left_join(x, y)` includes all observations in x, regardless of whether they match or not. This is the most commonly used join because it ensures that you don't lose observations from your primary table.




- `right_join(x, y)` includes all observations in y. It's equivalent to `left_join(y, x)`, but the columns will be ordered differently.

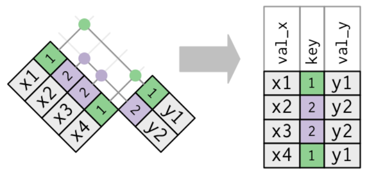



- `full_join()` includes all observations from x and y.

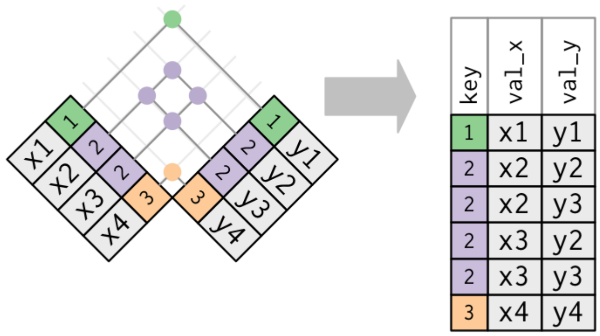



If there are duplicate keys, all combinations are returned.

**One to Many**




**Many to Many**




### 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 x 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       39.0  28.0  64.4
#> 2  2013     1     1     5 LGA    IAH   N24211  UA       39.9  25.0  54.8
#> 3  2013     1     1     5 JFK    MIA   N619AA  AA       39.0  27.0  61.6
#> 4  2013     1     1     5 JFK    BQN   N804JB  B6       39.0  27.0  61.6
#> 5  2013     1     1     6 LGA    ATL   N668DN  DL       39.9  25.0  54.8
#> 6  2013     1     1     5 EWR    ORD   N39463  UA       39.0  28.0  64.4
#> # ... with 3.368e+05 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 x 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       39.0  28.0  64.4
#> 2  2013     1     1     5 LGA    IAH   N24211  UA       39.9  25.0  54.8
#> 3  2013     1     1     5 JFK    MIA   N619AA  AA       39.0  27.0  61.6
#> 4  2013     1     1     5 JFK    BQN   N804JB  B6       39.0  27.0  61.6
```

```
#> 5  2013     1     1      6 LGA    ATL    N668DN  DL        39.9  25.0  54.8
#> 6  2013     1     1      5 EWR    ORD    N39463  UA        39.0  28.0  64.4
#> # ... with 3.368e+05 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 x 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>
#> # ... with 3.368e+05 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 x 16
#>   year.x month   day  hour origin dest  tailnum carrier year.y type
#>    <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>    <int> <chr>
#> 1   2013     1     1     5 EWR    IAH   N14228  UA        1999 Fixe~
#> 2   2013     1     1     5 LGA    IAH   N24211  UA        1998 Fixe~
#> 3   2013     1     1     5 JFK    MIA   N619AA  AA        1990 Fixe~
#> 4   2013     1     1     5 JFK    BQN   N804JB  B6        2012 Fixe~
#> 5   2013     1     1     6 LGA    ATL   N668DN  DL        1991 Fixe~
#> 6   2013     1     1     5 EWR    ORD   N39463  UA        2012 Fixe~
#> # ... with 3.368e+05 more rows, and 6 more variables: 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.

#### 2.1.1.1 Named Key Specification

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 x 15
#>    year month   day  hour origin dest  tailnum carrier name    lat   lon
#>   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr> <dbl> <dbl>
#> 1  2013     1     1     5 EWR    IAH   N14228  UA      Geor~  30.0 -95.3
#> 2  2013     1     1     5 LGA    IAH   N24211  UA      Geor~  30.0 -95.3
#> 3  2013     1     1     5 JFK    MIA   N619AA  AA      Miam~  25.8 -80.3
#> 4  2013     1     1     5 JFK    BQN   N804JB  B6      <NA>    NA    NA
```

```
#> 5  2013     1     1      6 LGA    ATL   N668DN  DL      Hart~  33.6 -84.4
#> 6  2013     1     1      5 EWR    ORD   N39463  UA      Chic~  42.0 -87.9
#> # ... with 3.368e+05 more rows, and 4 more variables: alt <int>, tz <dbl>,
#> #   dst <chr>, tzone <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 x 15
#>    year month   day  hour origin dest  tailnum carrier name    lat   lon
#>   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr> <dbl> <dbl>
#> 1  2013     1     1      5 EWR    IAH   N14228  UA      Newa~  40.7 -74.2
#> 2  2013     1     1      5 LGA    IAH   N24211  UA      La G~  40.8 -73.9
#> 3  2013     1     1      5 JFK    MIA   N619AA  AA      John~  40.6 -73.8
#> 4  2013     1     1      5 JFK    BQN   N804JB  B6      John~  40.6 -73.8
#> 5  2013     1     1      6 LGA    ATL   N668DN  DL      La G~  40.8 -73.9
#> 6  2013     1     1      5 EWR    ORD   N39463  UA      Newa~  40.7 -74.2
#> # ... with 3.368e+05 more rows, and 4 more variables: alt <int>, tz <dbl>,
#> #   dst <chr>, tzone <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 about the map components later in the course).

```
#-- Get new table of avg delay with airport lon/lat coordinates
dest_delays =
          flights %>%
          group_by(dest) %>%
          summarize(avg.delay = mean(arr_delay, na.rm=TRUE),
                    n.flights = n()) %>%
          inner_join(airports, by=c('dest' = 'faa'))
```

```
#-- Plot the aiports: color by average delay, size by number of flights
ggplot(dest_delays, aes(x=lon, y=lat, size=n.flights, color=avg.delay)) +
  geom_point() +
  scale_size_area() +
  scale_color_gradient2(low='#91bfdb', mid='#ffffbf', high='#fc8d59') +  # color gradient
  borders("state") +                                 # add map outline
  coord_quickmap(xlim=c(-125, -68), ylim=c(25, 50)) # mainland US only
```

2. We saw that MLB (baseball) players were more likely to be born in some months than others. But what about a player's name? Do MLB baseball players have unusual names?
   - The `babynames` package has a `babynames` dataset that gives popularity of US (first) names by year.
   - Calculate the proportion of names of MLB players for each year.
   - Join the baseball and babyname tables to compare the proportions.
   - Note the largest anomalies.

3. Is there a relationship between the age of a plane and its *average* delay?

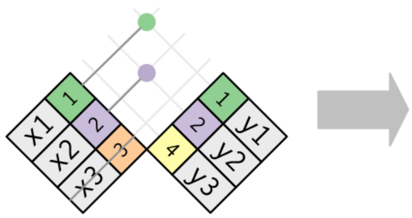
4. What weather conditions make it more likely to see a delay? Find the relationship between departure delays (`dep_delay`) and the weather variables at the origin (`dest`).

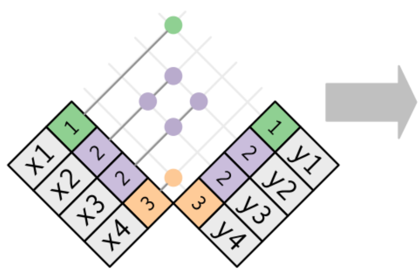
### 2.2 Filtering Joins (R4DS 13.5)

Filtering joins match observations in the same way as mutating joins, but affect the observations (rows), 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 **don't have** a match in y.

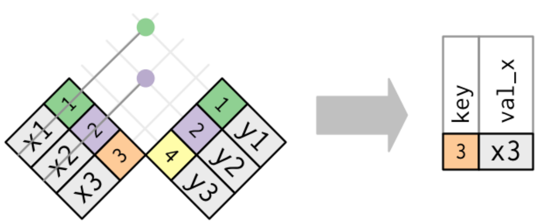
A semi-join connects two tables like a mutating join, but instead of adding new columns, only keeps the rows in `x` that have a match in `y`.




An anti-join is the reverse, it keeps the rows in `x` that do **not** have a match in `y`.

```
knitr::include_graphics("figs/join-anti.png")
```




#### 2.2.1 Your Turn: Joins

**Your Turn #3 : Joins**

1. What does `anti_join(flights, airports, by = c("dest" = "faa"))` tell you? What does `anti_join(airports, flights, by = c("faa" = "dest"))` tell you?
2. 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)` |

| dplyr | SQL |
|---|---|
| `semi_join()` | `SELECT * FROM x WHERE EXISTS (SELECT 1 FROM y WHERE x.a = y.a)` |
| `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.