Two-table verbs

It's rare that a data analysis involves only a single table of data. In practice, you'll normally have many tables that contribute to an analysis, and you need flexible tools to combine them. In dplyr, there are three families of verbs that work with two tables at a time:

- Mutating joins, which add new variables to one table from matching rows in another.
- Filtering joins, which filter observations from one table based on whether or not they match an observation in the other table.
- Set operations, which combine the observations in the data sets as if they were set elements.

(This discussion assumes that you have tidy data, where the rows are observations and the columns are variables. If you're not familiar with that framework, I'd recommend reading up on it first.)

All two-table verbs work similarly. The first two arguments are `x` and `y`, and provide the tables to combine. The output is always a new table with the same type as `x`.

Mutating joins

Mutating joins allow you to combine variables from multiple tables. For example, consider the flights and airlines data from the nycflights13 package. In one table we have flight information with an abbreviation for carrier, and in another we have a mapping between abbreviations and full names. You can use a join to add the carrier names to the flight data:

```
library(nycflights13)
# Drop unimportant variables so it's easier to understand the join results.
flights2 <- flights %>% select(year:day, hour, origin, dest, tailnum, carrier)

flights2 %>%
  left_join(airlines)
#> Joining with `by = join_by(carrier)`
#> # A tibble: 336,776 × 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 Inc.
#> 2  2013     1     1     5 LGA    IAH   N24211  UA      United Air Lines Inc.
#> 3  2013     1     1     5 JFK    MIA   N619AA  AA      American Airlines Inc.
#> 4  2013     1     1     5 JFK    BQN   N804JB  B6      JetBlue Airways
#> 5  2013     1     1     6 LGA    ATL   N668DN  DL      Delta Air Lines Inc.
#> # ℹ 336,771 more rows
```

Controlling how the tables are matched

As well as `x` and `y`, each mutating join takes an argument `by` that controls which variables are used to match observations in the two tables. There are a few ways to specify it, as I illustrate below with various tables from nycflights13:

- `NULL`, the default. dplyr will will use all variables that appear in both tables, a **natural** join. For example, the flights and weather tables match on their common variables: year, month, day, hour and origin.

```
flights2 %>% left_join(weather)
#> Joining with `by = join_by(year, month, day, hour, origin)`
#> # A tibble: 336,776 × 18
#>    year month   day  hour origin dest  tailnum carrier  temp  dewp humid
#>   <int> <int> <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
#> # ℹ 336,771 more rows
#> # ℹ 7 more variables: wind_dir <dbl>, wind_speed <dbl>, wind_gust <dbl>,
#> #   precip <dbl>, pressure <dbl>, visib <dbl>, time_hour <dttm>
```

- A character vector, `by = "x"`. Like a natural join, but uses only some of the common variables. For example, `flights` and `planes` have `year` columns, but they mean different things so we only want to join by `tailnum`.

```
flights2 %>% left_join(planes, by = "tailnum")
#> # A tibble: 336,776 × 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 Fixed wing multi…
#> 2   2013     1     1     5 LGA    IAH   N24211  UA        1998 Fixed wing multi…
#> 3   2013     1     1     5 JFK    MIA   N619AA  AA        1990 Fixed wing multi…
#> 4   2013     1     1     5 JFK    BQN   N804JB  B6        2012 Fixed wing multi…
#> 5   2013     1     1     6 LGA    ATL   N668DN  DL        1991 Fixed wing multi…
#> # ℹ 336,771 more rows
#> # ℹ 6 more variables: manufacturer <chr>, model <chr>, engines <int>,
#> #   seats <int>, speed <int>, engine <chr>
```

  Note that the year columns in the output are disambiguated with a suffix.

- A named character vector: `by = c("x" = "a")`. This will match variable `x` in table `x` to variable `a` in table `y`. The variables from use will be used in the output.

  Each flight has an origin and destination `airport`, so we need to specify which one we want to join to:

```
flights2 %>% left_join(airports, c("dest" = "faa"))
#> # A tibble: 336,776 × 15
#>    year month   day  hour origin dest  tailnum carrier name      lat   lon   alt
#>   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr>   <dbl> <dbl> <dbl>
#> 1  2013     1     1     5 EWR    IAH   N14228  UA      George…  30.0 -95.3    97
#> 2  2013     1     1     5 LGA    IAH   N24211  UA      George…  30.0 -95.3    97
#> 3  2013     1     1     5 JFK    MIA   N619AA  AA      Miami …  25.8 -80.3     8
#> 4  2013     1     1     5 JFK    BQN   N804JB  B6      <NA>     NA    NA      NA
#> 5  2013     1     1     6 LGA    ATL   N668DN  DL      Hartsf…  33.6 -84.4  1026
#> # ℹ 336,771 more rows
```

```
#> # ℹ 3 more variables: tz <dbl>, dst <chr>, tzone <chr>
flights2 %>% left_join(airports, c("origin" = "faa"))
#> # A tibble: 336,776 × 15
#>    year month   day  hour origin dest  tailnum carrier name      lat   lon   alt
#>   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr>   <dbl> <dbl> <dbl>
#> 1  2013     1     1     5 EWR    IAH   N14228  UA      Newark… 40.7 -74.2    18
#> 2  2013     1     1     5 LGA    IAH   N24211  UA      La Gua… 40.8 -73.9    22
#> 3  2013     1     1     5 JFK    MIA   N619AA  AA      John F… 40.6 -73.8    13
#> 4  2013     1     1     5 JFK    BQN   N804JB  B6      John F… 40.6 -73.8    13
#> 5  2013     1     1     6 LGA    ATL   N668DN  DL      La Gua… 40.8 -73.9    22
#> # ℹ 336,771 more rows
#> # ℹ 3 more variables: tz <dbl>, dst <chr>, tzone <chr>
```

Types of join

There are four types of mutating join, which differ in their behaviour when a match is not found. We'll illustrate each with a simple example:

```
df1 <- tibble(x = c(1, 2), y = 2:1)
df2 <- tibble(x = c(3, 1), a = 10, b = "a")
```

- `inner_join(x, y)` only includes observations that match in both `x` and `y`.

```
df1 %>% inner_join(df2) %>% knitr::kable()
#> Joining with `by = join_by(x)`
```

x	y	a	b
1	2	10	a

- `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.

```
df1 %>% left_join(df2)
#> Joining with `by = join_by(x)`
#> # A tibble: 2 × 4
#>       x     y     a b
#>   <dbl> <int> <dbl> <chr>
#> 1     1     2    10 a
#> 2     2     1    NA <NA>
```

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

```
df1 %>% right_join(df2)
#> Joining with `by = join_by(x)`
#> # A tibble: 2 × 4
#>       x     y     a b
```

```
#>   <dbl> <int> <dbl> <chr>
#> 1     1     2    10 a
#> 2     3    NA    10 a
df2 %>% left_join(df1)
#> Joining with `by = join_by(x)`
#> # A tibble: 2 × 4
#>       x     a b         y
#>   <dbl> <dbl> <chr> <int>
#> 1     3    10 a        NA
#> 2     1    10 a         2
```

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

```
df1 %>% full_join(df2)
#> Joining with `by = join_by(x)`
#> # A tibble: 3 × 4
#>       x     y     a b
#>   <dbl> <int> <dbl> <chr>
#> 1     1     2    10 a
#> 2     2     1    NA <NA>
#> 3     3    NA    10 a
```

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.

Observations

While mutating joins are primarily used to add new variables, they can also generate new observations. If a match is not unique, a join will add all possible combinations (the Cartesian product) of the matching observations:

```
df1 <- tibble(x = c(1, 1, 2), y = 1:3)
df2 <- tibble(x = c(1, 1, 2), z = c("a", "b", "a"))

df1 %>% left_join(df2)
#> Joining with `by = join_by(x)`
#> Warning in left_join(., df2): Detected an unexpected many-to-many relationship between
`x` and `y`.
#> ℹ Row 1 of `x` matches multiple rows in `y`.
#> ℹ Row 1 of `y` matches multiple rows in `x`.
#> ℹ If a many-to-many relationship is expected, set `relationship =
#>   "many-to-many"` to silence this warning.
#> # A tibble: 5 × 3
#>       x     y z
#>   <dbl> <int> <chr>
#> 1     1     1 a
#> 2     1     1 b
#> 3     1     2 a
#> 4     1     2 b
#> 5     2     3 a
```

Filtering joins

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`.

These are most useful for diagnosing join mismatches. For example, there are many flights in the nycflights13 dataset that don't have a matching tail number in the planes table:

```
library("nycflights13")
flights %>%
  anti_join(planes, by = "tailnum") %>%
  count(tailnum, sort = TRUE)
#> # A tibble: 722 × 2
#>   tailnum     n
#>   <chr>   <int>
#> 1 <NA>     2512
#> 2 N725MQ    575
#> 3 N722MQ    513
#> 4 N723MQ    507
#> 5 N713MQ    483
#> # ℹ 717 more rows
```

If you're worried about what observations your joins will match, start with a `semi_join()` or `anti_join()`. `semi_join()` and `anti_join()` never duplicate; they only ever remove observations.

```
df1 <- tibble(x = c(1, 1, 3, 4), y = 1:4)
df2 <- tibble(x = c(1, 1, 2), z = c("a", "b", "a"))

# Four rows to start with:
df1 %>% nrow()
#> [1] 4
# And we get four rows after the join
df1 %>% inner_join(df2, by = "x") %>% nrow()
#> Warning in inner_join(., df2, by = "x"): Detected an unexpected many-to-many relationship
between `x` and `y`.
#> ℹ Row 1 of `x` matches multiple rows in `y`.
#> ℹ Row 1 of `y` matches multiple rows in `x`.
#> ℹ If a many-to-many relationship is expected, set `relationship =
#>   "many-to-many"` to silence this warning.
#> [1] 4
# But only two rows actually match
df1 %>% semi_join(df2, by = "x") %>% nrow()
#> [1] 2
```

Set operations

The final type of two-table verb is set operations. These expect the `x` and `y` inputs to have the same variables, and treat 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`.

Given this simple data:

```
(df1 <- tibble(x = 1:2, y = c(1L, 1L)))
#> # A tibble: 2 × 2
#>       x     y
#>   <int> <int>
#> 1     1     1
#> 2     2     1
(df2 <- tibble(x = 1:2, y = 1:2))
#> # A tibble: 2 × 2
#>       x     y
#>   <int> <int>
#> 1     1     1
#> 2     2     2
```

The four possibilities are:

```
intersect(df1, df2)
#> # A tibble: 1 × 2
#>       x     y
#>   <int> <int>
#> 1     1     1
# Note that we get 3 rows, not 4
union(df1, df2)
#> # A tibble: 3 × 2
#>       x     y
#>   <int> <int>
#> 1     1     1
#> 2     2     1
#> 3     2     2
setdiff(df1, df2)
#> # A tibble: 1 × 2
#>       x     y
#>   <int> <int>
#> 1     2     1
setdiff(df2, df1)
#> # A tibble: 1 × 2
#>       x     y
#>   <int> <int>
#> 1     2     2
```

Multiple-table verbs

dplyr does not provide any functions for working with three or more tables. Instead use `purrr::reduce()` or `Reduce()`, as described in Advanced R, to iteratively combine the two-table verbs to handle as many tables as you need.