Joins

Grinnell College

February 10, 2026

Warm-up

You've got two lists:

- A list of people who RSVP'd to your party
- A list of people who showed up

Who do you email:

1. To say thank you for coming
2. To ask, "Hey, where were you?"
3. To ask, "Who are you and why were you at my house?"

Overview

- Keys
- Joins

Some Definitions

We will define a **table** as any tabular data made up of observations and values

A **database** is a thematically cohesive collection of tables that may be joined or linked together through identifiers

Keys

Most generally, a **key** is an identifier or set of identifiers for observations in a table

- **Primary Key** is a key associated with a table of interest that *uniquely identifies* each observation
- **Foreign Key** is a variable or set of variables that correspond to a primary key in a different table

Keys are the primary mechanism for relating different tables in a database

Key Considerations

- Single or compound keys
- Checking for uniqueness
- Anticipate conflicts from future entries in primary keys
- Use appropriate identifiers

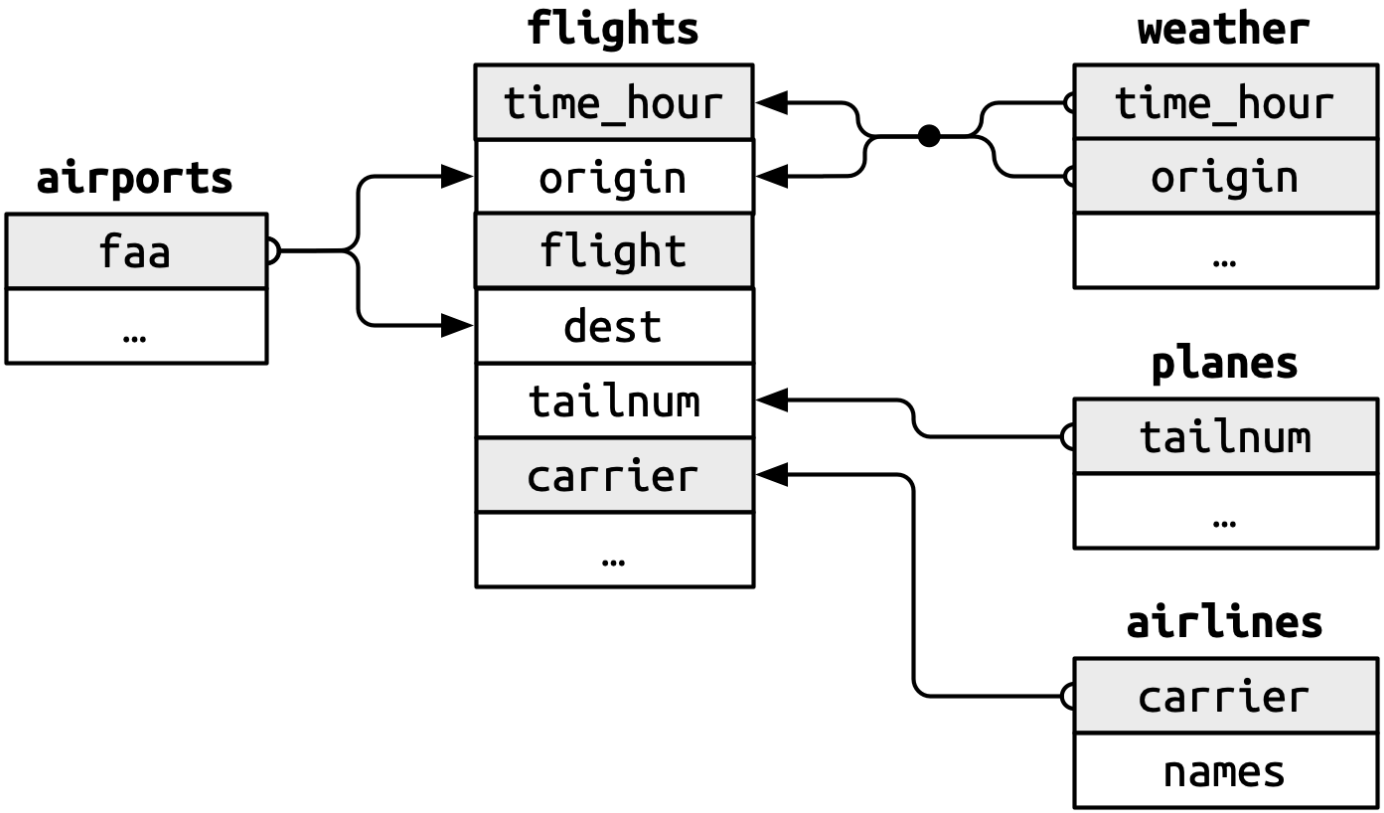
Primary Keys

```
> head(airlines)

  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.
```

Primary Keys

```
> head(airports[, c(1, 2, 3, 4)])

  faa   name                                lat   lon
  <chr> <chr>                             <dbl> <dbl>
1 04G   Lansdowne Airport                  41.1 -80.6
2 06A   Moton Field Municipal Airport      32.5 -85.7
3 06C   Schaumburg Regional                42.0 -88.1
4 06N   Randall Airport                    41.4 -74.4
5 09J   Jekyll Island Airport              31.1 -81.4
6 0A9   Elizabethton Municipal Airport     36.4 -82.2
```

Key Relations



Gray boxes represent primary keys

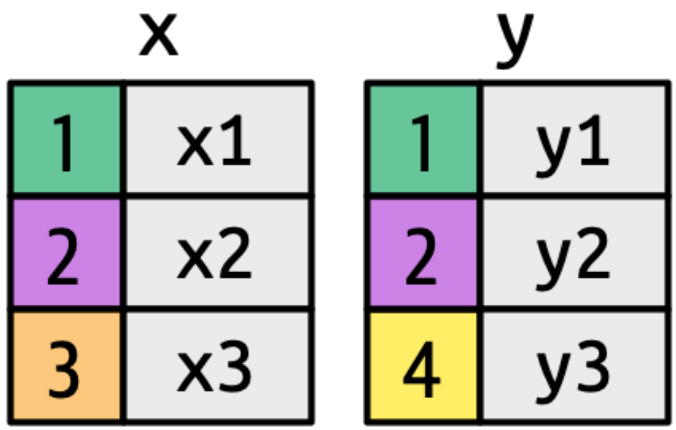
A Note on Notation

In virtually all of our examples, we'll assume we are working with two datasets or tables, X and Y

X will always be our primary table (with the primary key) and will serve as the first argument in any functions of the form $f(X, Y)$

This is especially important as most joins are *not* commutative

Visualizing Joins



Types of Joins

1. Mutating Joins
   - Defined by combination of different datasets
   - Typically used to add variables or metadata
   - Left, right, inner, full
2. Filtering Joins
   - Characterized by filtering of datasets
   - New variables are not added to existing
   - Semi, anti

Mutating Joins

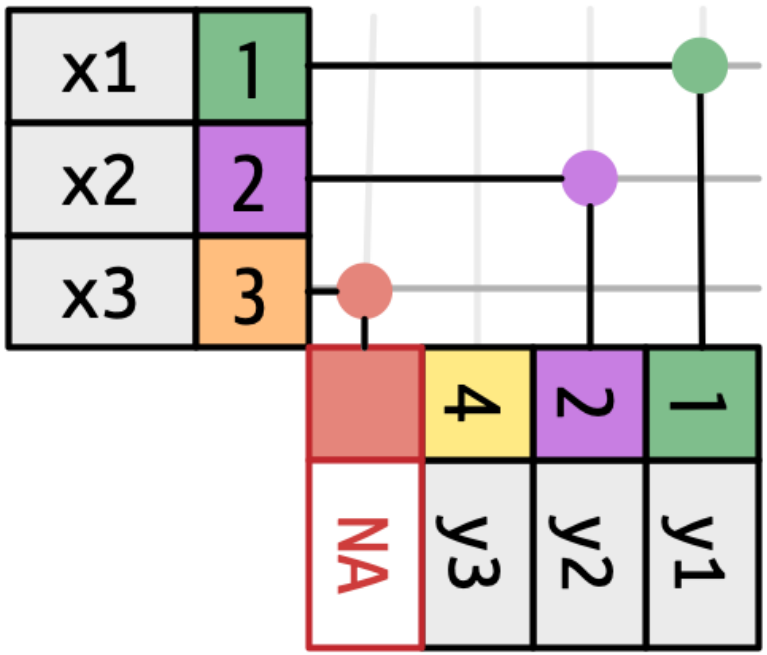
Left (and Right) Joins

The primary purpose of a left join is to preserve a primary table X while adding metadata from Y

- Retains all rows of X
- Adds corresponding variables from Y
- Adds `NA` to rows of Y when missing
- Example: bringing in plane specific data to the table of all flights

Left Joins

```
left_join(x, y)
```



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

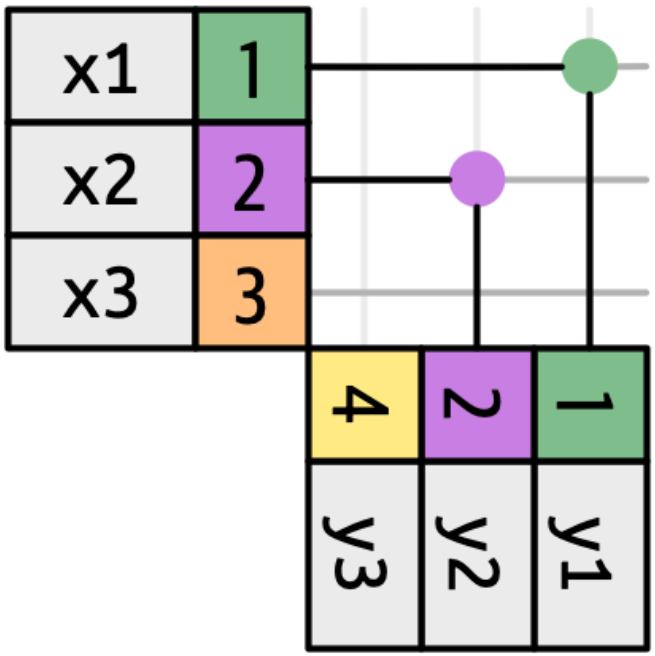
Inner Joins

Inner joins are primary used for finding complete records

- Retains only rows in X that have a match in Y
- This join is commutative
- Example: A table of all enrolled Grinnell students inner joined with all STEM courses offered Spring 2026 will return a list of all students enrolled in a science class (with class information)

Inner Joins

```
inner_join(x, y)
```



key	val_x	val_y
1	x1	y1
2	x2	y2

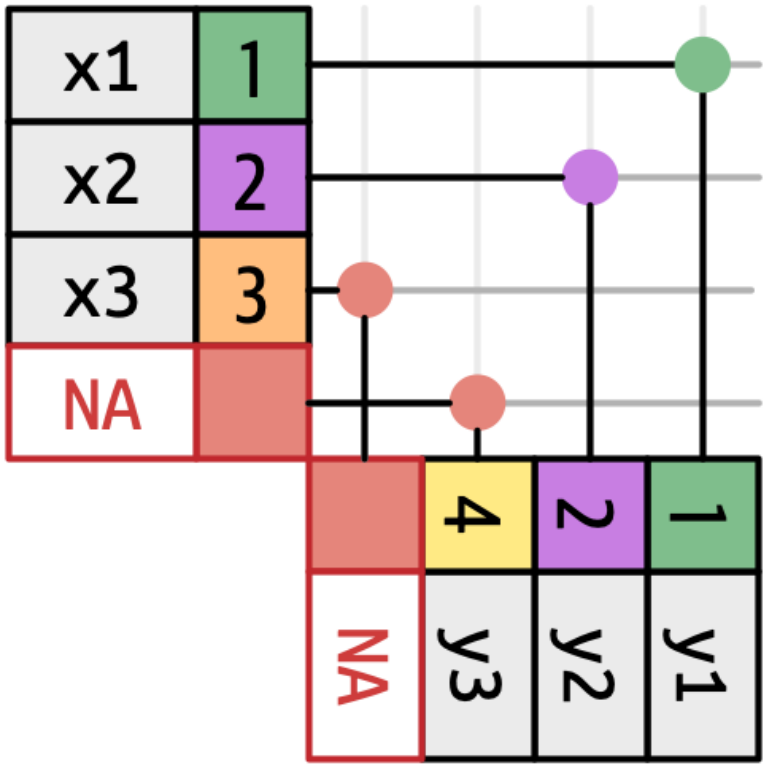
Outer/Full Joins

Outer joins are primarily used as a diagnostic tool

- Retains all rows from X and Y
- Adds `NA` in either when missing
- Quickly shows which tables have missing entries
- Example: Merging a list of all MAP students with all MAP directors will show if there are any students listed as having a MAP without and advisor or if any advisors are paid for a MAP but have no students

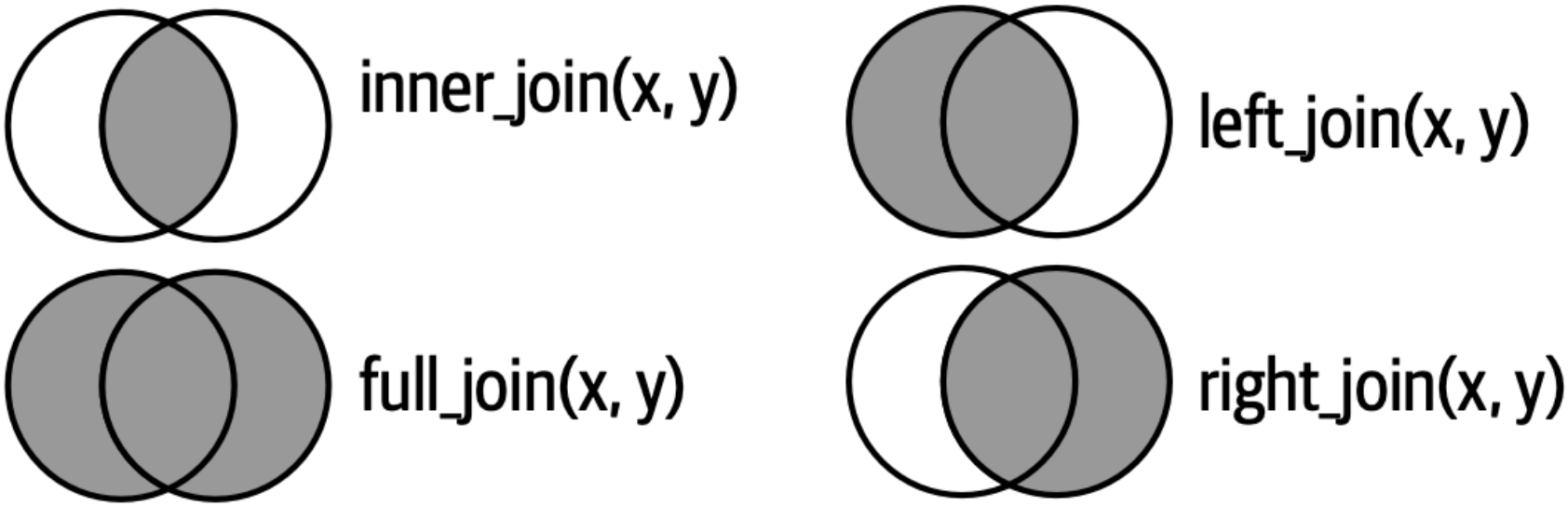
Full Joins

```
full_join(x, y)
```



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

Mutating Joins



Filtering Joins

Filtering Joins

In all cases, filtering joins are characterized by both removing rows from X and adding no new data from Y

Essentially we ask ourselves, am I filtering on a logical condition (for some purpose) or am I using this join as a diagnostic tool?

Both semi_join(x,y) and anti_join(x,y) are basically `dplyr::filter()` where the filtering criteria comes from a separate table

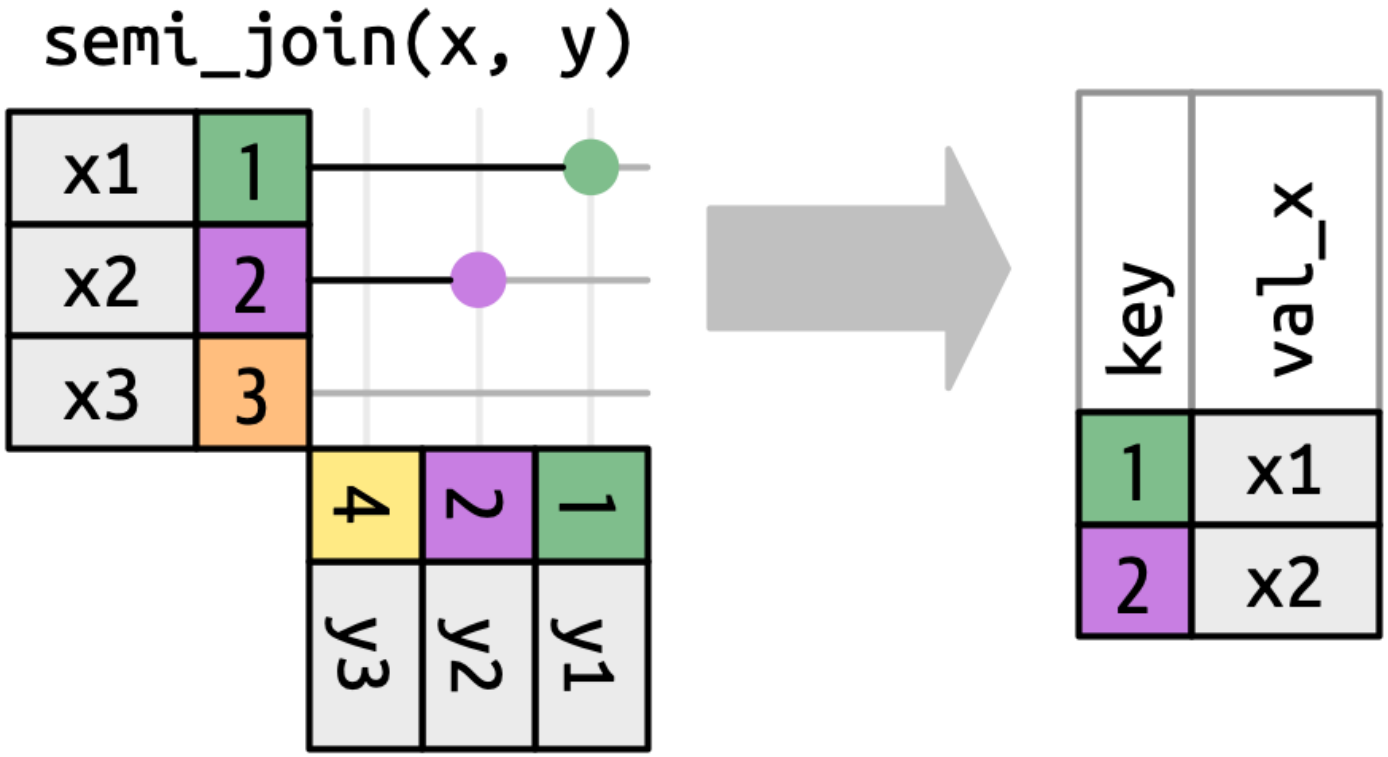
Semi Joins

Semi joins are used for keeping observations from one table that appear in another

Basically `semi_join(X, Y)` is equivalent to `dplyr::filter(X, key %in% Y$key)`

Example: From my list of customers, which ones have an order placed in the last 2 months?

Mutating Joins



Anti Joins

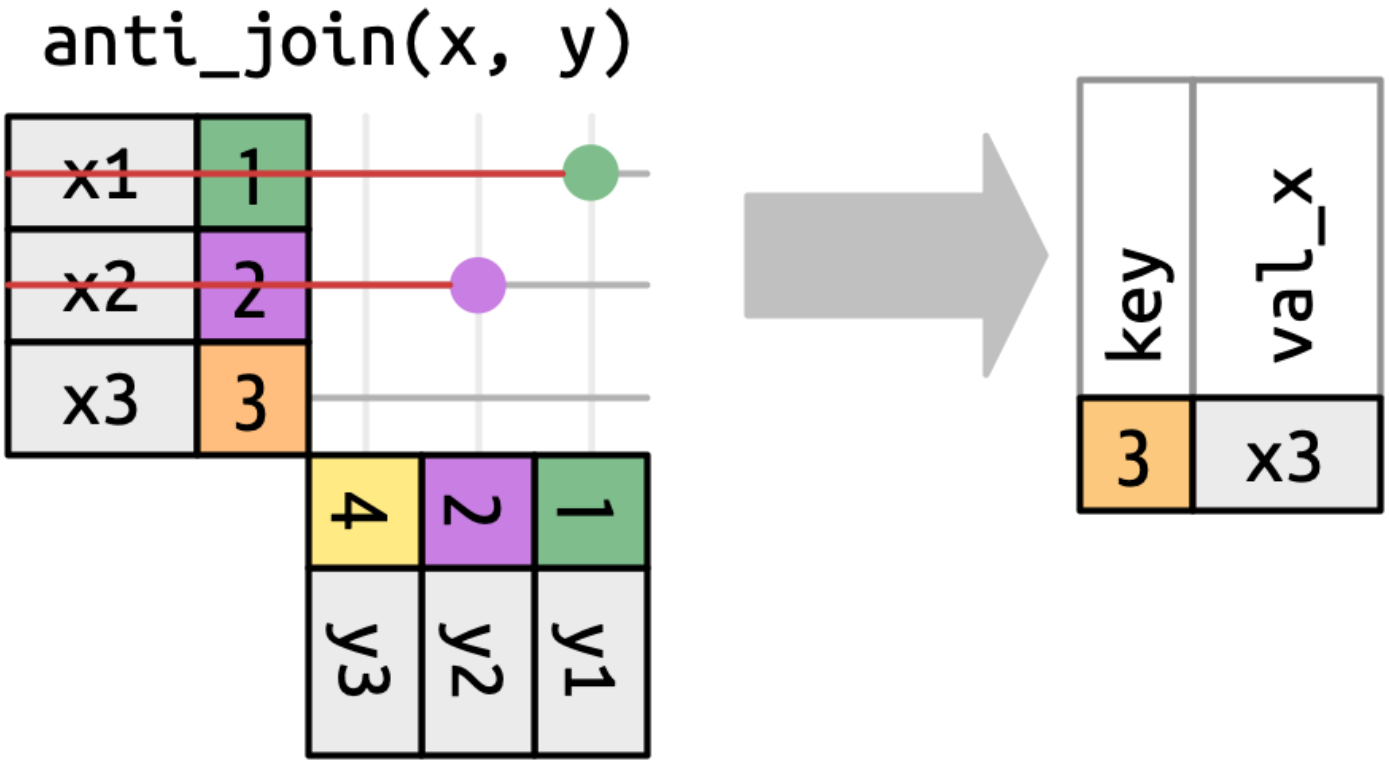
Almost exclusively a diagnostic tool for data quality

`anti_join(X, Y)` is equivalent to
`dplyr::filter(X, !(key %in% Y$key))`

Example: anti joining a list of Grinnell students with Spring 2026 enrollment will return a list of any students not enrolled in classes this semester

This is an example of *implicit* missing values; no `NA`s are returned

Mutating Joins



How do Joins Work?

- Equijoins vs nonequisjoins
- `by = "stuff"` vs `by = join_by(stuff)`
- `by = c("a" = "b")` vs `join_by(x == y)`
- Tidy selection
- Key name confusion
- What to retain