

8.1 dplyr: Keys and Mutating Joins

A primary key identifies each row. A foreign key points at that key in another table. `inner_join()` and `left_join()` add columns.

A primary key identifies each row of its own table. A foreign key points at a primary key in another table. `inner_join()` and `left_join()` add columns from that second table.

1. Several tables, one system

One table is rarely the whole story. A relational dataset splits the facts across tables. A **key** is the column, or the set of columns, that links them.

`nycflights13` is five tables about the same flights. Install the package once, then:

```
library(tidyverse)
library(nycflights13)

flights
planes
airlines
airports
weather
```

- `flights` — one row per flight out of NYC in 2013
- `planes` — one row per tail number
- `airlines` — carrier code and full name
- `airports` — one row per FAA code
- `weather` — hourly weather at the three NYC airports

Primary key

A **primary key** uniquely identifies each row in its own table.

- Every value is unique. There are no duplicates.
- It is never missing.
- It should not change over time.

`planes$tailnum`, `airlines$carrier`, and `airports$faa` are primary keys. The value picks out one row.

A name is a poor primary key. Two students can share one.

```
students <- tribble(
  ~student_id, ~name, ~major,
  1,           "Ana",  "Stats",
  2,           "Ben",  "Biology",
  3,           "Chloe", "Stats"
)

students
```

`student_id` is the primary key of `students`.

Foreign key

A **foreign key** is a column in one table that refers to the primary key of another table. That column is how you connect the tables.

```
enrollments <- tribble(
  ~enrollment_id, ~student_id, ~course, ~grade,
  101,            1,           "STAT101", "A",
  102,            1,           "BI0200",  "B",
  103,            3,           "STAT101", "A"
)

enrollments
```

`enrollment_id` is the primary key of `enrollments`. `student_id` is a foreign key. It points at `students$student_id`.

- Values can repeat. Ana has two enrollments, so `1` appears twice.
- Every value should exist in the referenced primary key. That rule is referential integrity. There is no row for student 99, because `students` has no student 99.
- A foreign key can be missing when the relationship is optional.

In `nycflights13` the foreign keys on `flights` are:

- `carrier` points at `airlines$carrier`. Every carrier is in `airlines`.
- `origin` points at `airports$faa`. Every origin is in `airports`.
- `dest` points at `airports$faa`. Four codes are not: `BQN`, `SJU`, `STT`, and `PSE`.
- `tailnum` points at `planes$tailnum`. Some flights have a missing tail number. Some tail numbers are not in `planes`.

The package stores the links. It does not enforce them. A later join is where the gaps show up.

Why split the tables

Ana's name and major are stored once, not copied onto every enrollment. A new major is one edit. You join when a question needs columns from both tables.

Relationships

- **One-to-many.** One student has many enrollments. The foreign key sits on the many side. One plane has many flights, so the foreign key is `flights$tailnum`.
- **Many-to-many.** Students take many courses, and a course has many students. Store that with a linking table. `enrollments` is that table: `student_id` refers to `students`, and `course` refers to a `courses` table whose primary key is the course id. Each foreign key is then one-to-many. Joining two tables that both repeat the key is a different problem, and lecture 8.2 treats it as a bad key.
- **One-to-one.** Each row in one table matches at most one row in the other.

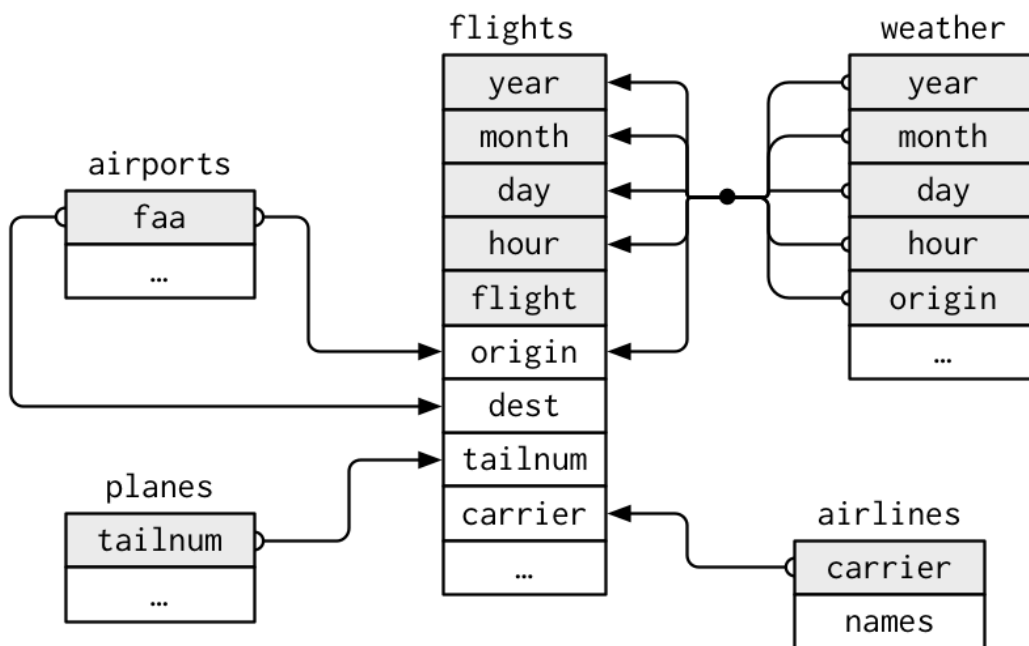
Composite key

Sometimes no single column is unique, and a combination is. That combination is a **composite primary key**.

`weather` is meant to be one row per airport-hour. The columns are `origin`, `year`, `month`, `day`, and `hour` together.

In `enrollments`, `(student_id, course)` could be the primary key if a student takes a given course only once. `enrollment_id` is a surrogate: an id you add so each row has a stable unique value.

`UCBAdmissions`, from the Week 6 mosaic, is already one table of counts: Admit, Gender, Dept, and `n`. Those facts could live in three tables instead. `applicants` would use `applicant_id` as its primary key and store gender. `departments` would use `dept_id` as its primary key. `applications` would store `applicant_id` and `dept_id` as foreign keys, plus the admit decision. Joining on those keys is what produces the Gender × Dept × Admit counts. The built-in table is already the counts, so the mosaic weights by `n` and does not join.



2. Check the keys

`count()` then `filter(n > 1)` finds duplicate keys. Zero rows means the columns are unique. Also check that a primary key has no missing values. A foreign key may repeat, and it may be missing when the link is optional.

```
students |>
  count(student_id) |>
  filter(n > 1)

students |>
  filter(is.na(student_id))

students |>
  left_join(enrollments, by = "student_id")

planes |>
  count(tailnum) |>
  filter(n > 1)

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

flights |>
  count(year, month, day, flight, tailnum) |>
  filter(n > 1)
```

The `students` check returns 0 rows, and `student_id` has no missing values, so it is a valid primary key. The join repeats Ana once per enrollment. That repeat is correct: the foreign key repeats, and the primary key does not.

If `count()` had returned rows, `student_id` would not be a primary key. The join would duplicate Ana for the wrong reason. That is a common silent error.

`planes$tailnum` passes the same check. `weather` does not. Hour 1 on November 3, 2013 appears twice at each NYC airport, because daylight saving time repeated that hour. Those five columns are still the intended composite key. The three extra rows are the exception.

The `flights` combination is not a key either. Those rows are different flights that share `year`, `month`, `day`, `flight`, and `tailnum`. On June 8, flight 2269 with tail number `N487WN` is both an LGA–STL departure and an EWR–DEN departure. When `tailnum` is missing, two unrelated flights collapse to the same key. `flights` has no natural primary key. Add a surrogate if you need one:

```
flights |>
  mutate(id = row_number())
```

3. Mutating joins add columns

The first argument is `x`, the second is `y`. The join matches rows by key and pastes columns from `y` onto `x`.

- `inner_join(x, y)` — only keys in both
- `left_join(x, y)` — every row of `x` (the default you will use)
- `right_join(x, y)` — every row of `y`
- `full_join(x, y)` — every key in either

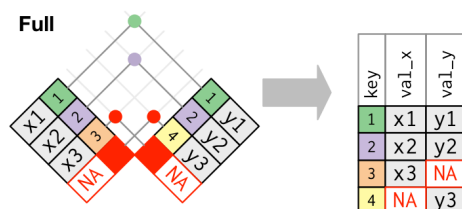
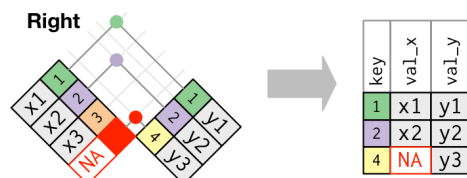
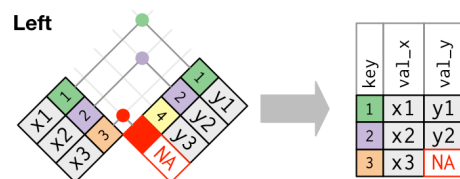
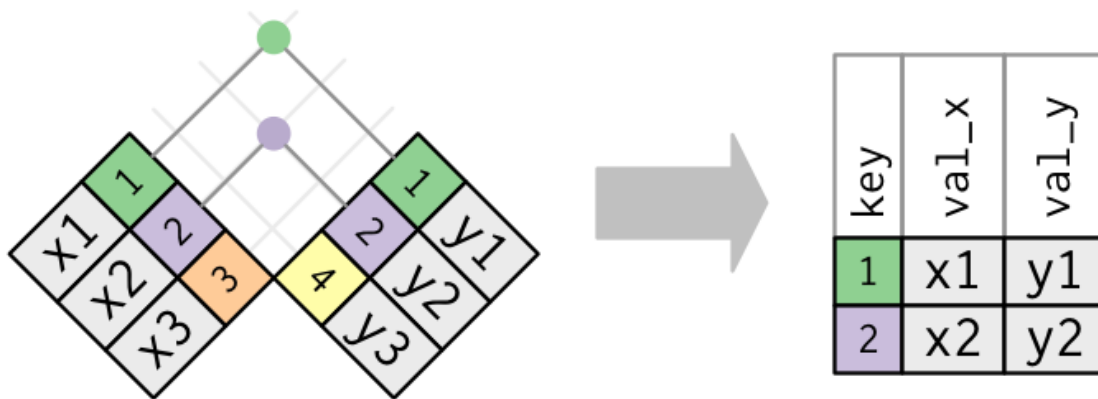
```

x <- tribble(
  ~key, ~val_x,
  1, "x1",
  2, "x2",
  3, "x3"
)
y <- tribble(
  ~key, ~val_y,
  1, "y1",
  2, "y2",
  4, "y4"
)

inner_join(x, y, by = "key")
left_join(x, y, by = "key")
right_join(x, y, by = "key")
full_join(x, y, by = "key")

```

Unmatched rows get **NA** in the new columns (`left_join` / `right_join` / `full_join`).



`nest_join(x, y)` is the primitive: each row of `x` gets a list-column of the matching rows of `y`. The other joins are that idea plus `unnest()`. You will use `left_join()` day to day. Know `nest_join()` exists when you want the matches as a list instead of extra rows.

4. by

Same name in both tables: `by = "carrier"`. Different names: a named vector, left name first.

```
flights |>
  left_join(airlines, by = "carrier")

flights |>
  left_join(airports, by = c("dest" = "faa"))
```

Several keys: `by = c("origin", "year", "month", "day", "hour")` for weather. If you omit `by`, `dplyr` uses every shared name and tells you which.

When both tables have a `year` (flights and planes), the join suffixes the clash: `year.x` and `year.y`, or set `suffix`. Safer: rename before you join.

```
flights |>
  left_join(
    planes |> select(tailnum, plane_year = year),
    by = "tailnum"
  )
```

5. Practice

1. Add the airline name to every flight.
2. Keep only flights that have a matching row in `planes`. How many flights are dropped?
3. Name a primary key for `ggplot2::diamonds`. If there is none, say so. Optional: the same question for `Lahman::Batting` or `babynames::babynames` if you install those packages.