Four joins, and the question each one answers

Inner, left, full and anti — chosen from what you need to be true of the result rather than from habit. Plus the row-count arithmetic that explains every fan-out you will ever see.

Pierre Robentz Cassion

Lesson 1 of 8

Joining and Reshaping Programme Data — Joins you can prove

What this lesson covers

- Pick the join from the sentence you are going to write
- The arithmetic, once, by hand
- The setup
- Left join: keep the left table, attach columns
- Inner join: and the rows it takes with it
- Full join: nothing is lost, and now you have two kinds of blank
- Anti-join: the check, not the join
- Joining on the wrong column
- What comes next

Pick the join from the sentence you are going to write

The sentence you will write	The join
"Attendance for every enrolled student"	left, register on the left
"Attendance for students we have both a record and a roster row for"	inner
"Everything from both sides, so nothing is lost"	full
"Students marked present who are not on the roster"	anti

The arithmetic, once, by hand

Left occurrences	Right occurrences	Rows emitted
1	1	1
60	1	60
60	2	120
50	3000	150,000

The arithmetic, once, by hand

- **Row count preserved** only when the right side is unique on the key. This is the sentence the whole lesson rests on.
- **Row count multiplied** when it is not — quietly, with no error, by a factor nobody chose.
- **Row count reduced** only by an inner join, or by a key that fails to match. A left join never removes a row, so a…

The setup — In Python

```
import pandas as pd

roster = pd.read_csv("school-roster-2024.v1.csv")
attendance = pd.read_csv("school-attendance-2024.v1.csv", parse_dates=["attendance_date"])

print(len(roster), roster["student_id"].nunique())
print(len(attendance))
```

The setup — In R

```
library(dplyr)
library(readr)

roster     <- read_csv("school-roster-2024.v1.csv")
attendance <- read_csv("school-attendance-2024.v1.csv")

c(rows = nrow(roster), students = n_distinct(roster$student_id))
nrow(attendance)
```

The setup — In Python

```
roster = roster.drop_duplicates(subset=["student_id"], keep="first")
```

The setup — In R

```
roster <- roster |> distinct(student_id, .keep_all = TRUE)
```

The setup

- Keeping the first row is a decision, not a default — For these two students it is the wrong one — one row records the...

Left join: keep the left table, attach columns — In Python

```
joined = attendance.merge(roster, on="student_id", how="left", validate="many_to_one")
print(len(attendance), "->", len(joined))
```

Left join: keep the left table, attach columns — In R

```
joined <- attendance |>
  left_join(roster, by = "student_id", relationship = "many-to-one")

cat(nrow(attendance), "->", nrow(joined), "\n")
```

Inner join: and the rows it takes with it — In Python

```
inner = attendance.merge(roster, on="student_id", how="inner")
print(len(inner), "rows;", len(attendance) - len(inner), "attendance rows dropped")
```

Inner join: and the rows it takes with it — In R

```r
inner <- attendance |> inner_join(roster, by = "student_id")
cat(nrow(inner), "rows;", nrow(attendance) - nrow(inner), "dropped\n")
```

Inner join: and the rows it takes with it

Use an inner join when you have already looked at what it removes. Using it *to* remove them is how the awkward rows get disposed of without a decision.

Full join: nothing is lost, and now you have two kinds of blank — In Python

```
full = attendance.merge(roster, on="student_id", how="outer", indicator=True)
print(full["_merge"].value_counts())
```

Full join: nothing is lost, and now you have two kinds of blank — In R

```
full <- attendance |> full_join(roster, by = "student_id")
```

Full join: nothing is lost, and now you have two kinds of blank — In R

```
full <- attendance |>
  mutate(in_attendance = TRUE) |>
  full_join(mutate(roster, in_roster = TRUE), by = "student_id") |>
  mutate(
    side = case_when(
      in_attendance & in_roster ~ "both",
      in_attendance             ~ "attendance only",
      TRUE                      ~ "roster only"
    )
  )
```

Anti-join: the check, not the join — In Python

```
in_roster = set(roster["student_id"])
orphans = attendance[~attendance["student_id"].isin(in_roster)]
never_seen = roster[~roster["student_id"].isin(set(attendance["student_id"]))]

print(len(orphans), "attendance rows with no roster entry")
print(len(never_seen), "enrolled students with no attendance row at all")
```

Anti-join: the check, not the join — In R

```r
orphans     <- attendance |> anti_join(roster, by = "student_id")
never_seen  <- roster |> anti_join(attendance, by = "student_id")

c(orphans = nrow(orphans), never_seen = nrow(never_seen))
```

Anti-join: the check, not the join

- **Attendance with no roster row** — someone is being recorded who is not enrolled. A data problem, or an enrolment list...
- **Roster rows with no attendance** — enrolled children never marked either way. In an education programme that is not a...

Joining on the wrong column — In Python

```
wrong = attendance.merge(roster, on="school_id", how="left")
print(f"{len(attendance):,} -> {len(wrong):,}")
```

Joining on the wrong column — In R

```
wrong <- attendance |> left_join(roster, by = "school_id")
format(nrow(wrong), big.mark = ",")
```

What comes next

- You have four joins and the arithmetic that governs them.

Read the full lesson, with runnable code

https://data-analysis.cassion.dev/courses/joining-and-reshaping/joining-01-the-four-joins