Quatre jointures, et la question à laquelle chacune répond

Interne, à gauche, complète et anti — choisies d'après ce qui doit être vrai du résultat plutôt que par habitude. Plus l'arithmétique des lignes qui explique tous les gonflements que vous rencontrerez.

Pierre Robentz Cassion

Leçon 1 sur 8

Jointures et restructuration des données de programme — Des jointures démontrables

Ce que couvre cette leçon

- Choisissez la jointure d'après la phrase que vous allez écrire
- L'arithmétique, une fois, à la main
- Le montage
- Jointure à gauche — garder la table de gauche, y accrocher des colonnes
- Jointure interne — et les lignes qu'elle emporte
- Jointure complète — rien n'est perdu, et vous avez désormais deux sortes de vide
- Anti-jointure — le contrôle, pas la jointure
- Joindre sur la mauvaise colonne
- La suite

Choisissez la jointure d'après la phrase que vous allez écrire

La phrase que vous écrirez	**La jointure**
« La présence de chaque élève inscrit »	à gauche, registre à gauche
« La présence des élèves ayant à la fois une trace et une ligne de liste »	interne
« Tout des deux côtés, pour ne rien perdre »	complète
« Les élèves marqués présents qui ne sont pas sur la liste »	anti

L'arithmétique, une fois, à la main

Occurrences à gauche	Occurrences à droite	Lignes émises
1	1	1
60	1	60
60	2	120
50	3000	150 000

L'arithmétique, une fois, à la main

- **Nombre de lignes préservé** seulement si le côté droit est unique sur la clé. C'est la phrase sur laquelle repose…
- **Nombre de lignes multiplié** dans le cas contraire — en silence, sans erreur, par un facteur que personne n'a choisi.
- **Nombre de lignes réduit** uniquement par une jointure interne, ou par une clé qui ne s'apparie pas. Une jointure à…

Le montage — En 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))
```

Le montage — En 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)
```

Le montage — En Python

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

Le montage — En R

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

Le montage

- Garder la première ligne est une décision, pas un réglage par défaut — Pour ces deux élèves, c'est la mauvaise — l'une...

Jointure à gauche — garder la table de gauche, y accrocher des colonnes — En Python

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

Jointure à gauche — garder la table de gauche, y accrocher des colonnes — En R

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

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

Jointure interne — et les lignes qu'elle emporte — En Python

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

Jointure interne — et les lignes qu'elle emporte — En R

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

Jointure interne — et les lignes qu'elle emporte

Utilisez une jointure interne quand vous avez déjà regardé ce qu'elle retire. L'utiliser *pour* les retirer est la manière dont les lignes gênantes sont évacuées sans décision.

Jointure complète — rien n'est perdu, et vous avez désormais deux sortes de vide — En Python

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

Jointure complète — rien n'est perdu, et vous avez désormais deux sortes de vide — En R

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

Jointure complète — rien n'est perdu, et vous avez désormais deux sortes de vide — En R

```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-jointure — le contrôle, pas la jointure — En 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-jointure — le contrôle, pas la jointure — En 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-jointure — le contrôle, pas la jointure

- **De la présence sans ligne de liste** — on enregistre quelqu'un qui n'est pas inscrit. Un problème de données, ou une…
- **Des lignes de liste sans aucune présence** — des enfants inscrits jamais marqués, ni dans un sens ni dans l'autre….

Joindre sur la mauvaise colonne — En Python

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

Joindre sur la mauvaise colonne — En R

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

La suite

- Vous disposez de quatre jointures et de l'arithmétique qui les gouverne.

Lire la leçon complète, avec le code exécutable

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