

Prouver que la jointure a fait ce que vous avez annoncé

Un tableau de rapprochement pour chaque jointure — lignes appariées, les deux comptes d'anti-jointure, le nombre de lignes avant et après — plus la collision de suffixes qui écrase une colonne sans rien dire.

Pierre Robentz Cassion

Leçon 2 sur 8

Jointures et restructuration des données de programme — Des jointures démontrables

Ce que couvre cette leçon

- Le contrôle doit coûter moins cher que le bug
- Déclarez la relation, toujours
- Le tableau de rapprochement
- Protégez le nombre de lignes quand vous voulez le préserver
- Le taux de non-appariement est un indicateur, pas une erreur
- Normalisez la clé avant d'accuser la jointure
- La collision de suffixes
- Des clés aux noms différents
- La suite

Le contrôle doit coûter moins cher que le bug

- Personne ne saute les contrôles de jointure en croyant les jointures sûres.

Déclarez la relation, toujours — En Python

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

Déclarez la relation, toujours — En R

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

Déclarez la relation, toujours

Ce que vous attendez	pandas validate=	dplyr relationship=
Une ligne de chaque côté	one_to_one	one-to-one
Plusieurs à gauche, une à droite	many_to_one	many-to-one
Une à gauche, plusieurs à droite	one_to_many	one-to-many
Réellement plusieurs à plusieurs	omettre	many-to-many

Déclarez la relation, toujours

- dplyr avertit d'un plusieurs-à-plusieurs inattendu même sans l'argument — pandas non

Le tableau de rapprochement — En Python (suite)

```
def reconcile(left, right, on, left_name="left", right_name="right"):
    left_keys = left[on].drop_duplicates()
    right_keys = right[on].drop_duplicates()
    merged = left_keys.merge(right_keys, on=on, how="outer", indicator=True)
    counts = merged["_merge"].value_counts()

    return pd.Series({
        f"{left_name} rows": len(left),
        f"{right_name} rows": len(right),
        "keys matched": int(counts.get("both", 0)),
        f"{left_name} only": int(counts.get("left_only", 0)),
        f"{right_name} only": int(counts.get("right_only", 0)),
        f"{left_name} unmatched %": round(
            100 * counts.get("left_only", 0) / max(len(left_keys), 1), 2
        ),
    })
```

Le tableau de rapprochement — En Python (suite)

```
print(reconcile(attendance, roster, ["student_id"], "attendance", "roster"))
```

Le tableau de rapprochement — En R (suite)

```
reconcile <- function(left, right, on, left_name = "left", right_name = "right") {  
  left_keys <- dplyr::distinct(dplyr::select(left, dplyr::all_of(on)))  
  right_keys <- dplyr::distinct(dplyr::select(right, dplyr::all_of(on)))  
  
  tibble::tibble(  
    measure = c(paste(left_name, "rows"), paste(right_name, "rows"),  
               "keys matched", paste(left_name, "only"), paste(right_name, "only")),  
    value = c(  
      nrow(left), nrow(right),  
      nrow(dplyr::inner_join(left_keys, right_keys, by = on)),  
      nrow(dplyr::anti_join(left_keys, right_keys, by = on)),  
      nrow(dplyr::anti_join(right_keys, left_keys, by = on))  
    )  
  )  
}
```

Le tableau de rapprochement — En R (suite)

```
reconcile(attendance, roster, "student_id", "attendance", "roster")
```

Le tableau de rapprochement

mesure	valeur
lignes de présence	70 245
lignes de liste	1 200
clés appariées	1 200
présence seulement	0
liste seulement	0

Protégez le nombre de lignes quand vous vouliez le préserver — En Python

```
def join_preserving(left, right, on, how="left", **kwargs):  
    before = len(left)  
    out = left.merge(right, on=on, how=how, **kwargs)  
    if len(out) != before:  
        raise ValueError(f"join changed row count: {before} -> {len(out)}")  
    return out
```

Protégez le nombre de lignes quand vous vouliez le préserver — En R

```
join_preserving <- function(left, right, by, ...) {  
  before <- nrow(left)  
  out <- dplyr::left_join(left, right, by = by, ...)  
  if (nrow(out) != before) {  
    stop(sprintf("join changed row count: %d -> %d", before, nrow(out)))  
  }  
  out  
}
```

Le taux de non-appariement est un indicateur, pas une erreur — En Python

```
summary = reconcile(distributions, registration, ["beneficiary_id"])
if summary["left unmatched %"] > 5:
    print(f"WARNING: {summary['left unmatched %']}% of distribution rows "
          "have no registration record")
```

Le taux de non-appariement est un indicateur, pas une erreur — En R

```
unmatched <- nrow(anti_join(distributions, registration, by = "beneficiary_id"))
share <- unmatched / nrow(distributions)
if (share > 0.05) warning(sprintf("%.1f%% of distributions have no registration", 100 * share
  ))
```

Normalisez la clé avant d'accuser la jointure

- **Le type.** Un code de formation sanitaire lu comme entier d'un côté et comme chaîne de l'autre. 1042 n'égale jamais. . .
- **Espaces et casse.** "FAC001 " et "fac001" font trois clés différentes.
- **Dates contre horodatages.** 2024-03-01 et 2024-03-01 00:00:00 se comparent égaux dans certaines bibliothèques et. . .

Normalisez la clé avant d'accuser la jointure — En Python

```
def key_clean(series):  
    return series.astype("string").str.strip().str.upper()  
  
for frame in (registers, aggregates):  
    frame["facility_id"] = key_clean(frame["facility_id"])
```

Normalisez la clé avant d'accuser la jointure — En R

```
key_clean <- function(x) toupper(stringr::str_squish(as.character(x)))

registers <- registers |> mutate(facility_id = key_clean(facility_id))
aggregates <- aggregates |> mutate(facility_id = key_clean(facility_id))
```

Normalisez la clé avant d'accuser la jointure

- Faites-le sur les deux côtés dans la même fonction — pour qu'ils ne puissent pas diverger

```
merged = registers.merge(aggregates, on="facility_id", suffixes=("_register", "_dhis2"))
```

```
merged <- registers |>  
  left_join(aggregates, by = "facility_id", suffix = c("_register", "_dhis2"))
```

La collision de suffixes — En Python

```
overlap = (set(registers.columns) & set(aggregates.columns)) - {"facility_id"}  
print("columns in both:", sorted(overlap))
```

```
setdiff(intersect(names(registers), names(aggregates)), "facility_id")
```

Des clés aux noms différents — En Python

```
merged = cases.merge(  
    facilities, left_on="site_code", right_on="facility_id", how="left",  
    validate="many_to_one",  
)
```

Des clés aux noms différents — En R

```
merged <- cases |>  
  left_join(facilities, by = c("site_code" = "facility_id"),  
            relationship = "many-to-one")
```

- Tous les contrôles vus jusqu'ici supposent que vous savez ce qu'est une ligne de chaque table.

Lire la leçon complète, avec le code exécutable

[https://data-analysis.cassion.dev/fr/cours/joining-and-reshaping/
joining-02-proving-the-join](https://data-analysis.cassion.dev/fr/cours/joining-and-reshaping/joining-02-proving-the-join)