# Jointures entre tables

Bases de données

## 1 Pourquoi les jointures ?

Le modèle relationnel répartit les données dans plusieurs tables pour éviter la redondance. Mais quand on souhaite combiner des informations provenant de tables différentes, il faut les **relier** : c'est le rôle de la **jointure**.

## 2 Tables d'exemple

On considère une base de données de gestion scolaire avec trois tables.

**Eleves :**

| id_eleve | nom | prenom | id_classe |
|---|---|---|---|
| 1 | Dupont | Marie | 101 |
| 2 | Martin | Lucas | 102 |
| 3 | Leroy | Emma | 101 |
| 4 | Petit | Hugo | 102 |

**Classes :**

| id_classe | nom_classe | salle |
|---|---|---|
| 101 | Terminale A | B204 |
| 102 | Terminale B | C112 |

**Matieres :**

| id_matiere | nom_matiere | #id_classe | enseignant |
|---|---|---|---|
| 1 | NSI | 101 | M. Dupuis |
| 2 | Mathématiques | 101 | Mme Roche |
| 3 | NSI | 102 | M. Dupuis |
| 4 | Physique | 102 | M. Laurent |

L'attribut `id_classe` dans Eleves est une clé étrangère qui référence `Classes.id_classe`. C'est cet attribut commun qui permet de relier les deux tables.

## 3 Comprendre la jointure pas à pas

On souhaite afficher, pour chaque élève, son nom et le nom de sa classe.

**Étape 1 : identifier l'attribut commun.** `id_classe` est présent dans les deux tables.

**Étape 2 : faire correspondre les lignes.** Pour Marie Dupont, `id_classe = 101`. Dans Classes, la ligne 101 correspond à « Terminale A ». On associe donc Marie à « Terminale A ».

**Étape 3 : construire le résultat.**

| nom | prenom | nom_classe | salle |
|---|---|---|---|
| Dupont | Marie | Terminale A | B204 |
| Martin | Lucas | Terminale B | C112 |
| Leroy | Emma | Terminale A | B204 |
| Petit | Hugo | Terminale B | C112 |

## ④ Syntaxe SQL : JOIN … ON

C'est la syntaxe recommandée. Elle exprime clairement quelle table on rejoint et sur quelle condition.

```sql
SELECT Eleves.nom, Eleves.prenom, Classes.nom_classe
FROM Eleves
JOIN Classes ON Eleves.id_classe = Classes.id_classe;
```

Quand l'attribut de jointure porte **le même nom** dans les deux tables, on peut aussi écrire `USING`, suivi de ce nom entre parenthèses : l'égalité est alors sous-entendue, et la colonne `id_classe` n'apparaît qu'une fois dans le résultat.

```sql
SELECT nom, prenom, nom_classe
FROM Eleves
JOIN Classes USING (id_classe);
```

On peut ajouter un filtre avec `WHERE` :

```sql
SELECT Eleves.nom, Eleves.prenom
FROM Eleves
JOIN Classes ON Eleves.id_classe = Classes.id_classe
WHERE Classes.salle = 'B204';
```

Résultat : Dupont Marie, Leroy Emma.

## ⑤ Syntaxe SQL : WHERE (produit cartésien filtré)

Une autre syntaxe, plus ancienne, consiste à lister les tables dans `FROM` et à exprimer la condition de jointure dans `WHERE` :

```sql
SELECT Eleves.nom, Eleves.prenom, Classes.nom_classe
FROM Eleves, Classes
WHERE Eleves.id_classe = Classes.id_classe;
```

Le résultat est identique. Sans la condition `WHERE`, le SGBD calculerait le **produit cartésien** (toutes les combinaisons possibles de lignes des deux tables, soit $4 \times 2 = 8$ lignes), ce qui ne ferait pas sens.

Les deux syntaxes sont équivalentes, mais `JOIN … ON` est préférable car elle sépare clairement la condition de jointure des filtres.

## 6 Jointure sur trois tables

Pour combiner trois tables, on enchaîne les clauses `JOIN`.

```sql
SELECT Eleves.nom, Eleves.prenom,
       Matieres.nom_matiere, Matieres.enseignant
FROM Eleves
JOIN Classes ON Eleves.id_classe = Classes.id_classe
JOIN Matieres ON Classes.id_classe = Matieres.id_classe;
```

Résultat :

| nom | prenom | nom_matiere | enseignant |
|---|---|---|---|
| Dupont | Marie | NSI | M. Dupuis |
| Dupont | Marie | Mathématiques | Mme Roche |
| Leroy | Emma | NSI | M. Dupuis |
| Leroy | Emma | Mathématiques | Mme Roche |
| Martin | Lucas | NSI | M. Dupuis |
| Martin | Lucas | Physique | M. Laurent |
| Petit | Hugo | NSI | M. Dupuis |
| Petit | Hugo | Physique | M. Laurent |

On peut filtrer : « quels élèves ont M. Dupuis comme enseignant ? »

```sql
SELECT Eleves.nom, Eleves.prenom
FROM Eleves
JOIN Classes ON Eleves.id_classe = Classes.id_classe
JOIN Matieres ON Classes.id_classe = Matieres.id_classe
WHERE Matieres.enseignant = 'M. Dupuis';
```

Résultat : tous les élèves des deux classes (car M. Dupuis enseigne la NSI dans les deux classes).

## 7 Alias de table

Pour raccourcir l'écriture, on peut donner à chaque table un **alias**, c'est-à-dire un nom court, avec le mot-clé `AS` placé après le nom de la table dans `FROM` ou `JOIN`. L'alias remplace ensuite le nom de la table dans toute la requête : on écrit `E.nom` au lieu de `Eleves.nom`. Le mot `AS` peut être omis : `FROM Eleves E` a le même sens que `FROM Eleves AS E`.

```sql
SELECT E.nom, E.prenom, M.nom_matiere, M.enseignant
FROM Eleves AS E
JOIN Classes AS C ON E.id_classe = C.id_classe
JOIN Matieres AS M ON C.id_classe = M.id_classe;
```

Cette requête donne le même résultat que la jointure sur trois tables écrite plus haut.

## 8 Compter avec une jointure

Les fonctions d'agrégation fonctionnent aussi avec les jointures.

**Combien d'élèves sont en Terminale A ?**

```sql
SELECT COUNT(*)
FROM Eleves
JOIN Classes ON Eleves.id_classe = Classes.id_classe
WHERE Classes.nom_classe = 'Terminale A';
```

Résultat : 2 (Dupont Marie et Leroy Emma).

## 9 Synthèse

— Une **jointure** combine les lignes de deux (ou plusieurs) tables en se basant sur un attribut commun ;

— la syntaxe recommandée est `JOIN … ON` ;

— la syntaxe `FROM T1, T2 WHERE T1.clé = T2.clé` est équivalente mais moins explicite ;

— sans condition de jointure, on obtient le **produit cartésien** ;

— on peut enchaîner plusieurs `JOIN` pour relier trois tables ou plus ;

— les fonctions d'agrégation fonctionnent aussi avec les jointures.