Comprendre les jointures en SQL

Quand tu travailles avec une base de données relationnelle, les informations sont souvent éparpillées dans plusieurs tables. Par exemple, les informations sur les clients sont dans une table, les commandes dans une autre, et les produits encore dans une autre. À un moment, tu voudras **relier ces données** pour répondre à une question complète : « Qui a commandé quoi, et quand ? ». C'est là qu'interviennent les **jointures**.

niveau

✨ Introduction

Une **jointure** permet de combiner des lignes provenant de plusieurs tables grâce à un champ commun (souvent une clé étrangère). C'est un outil fondamental en SQL pour naviguer dans une base de données bien conçue, en liant logiquement les entités.

🌟 Objectif du cours

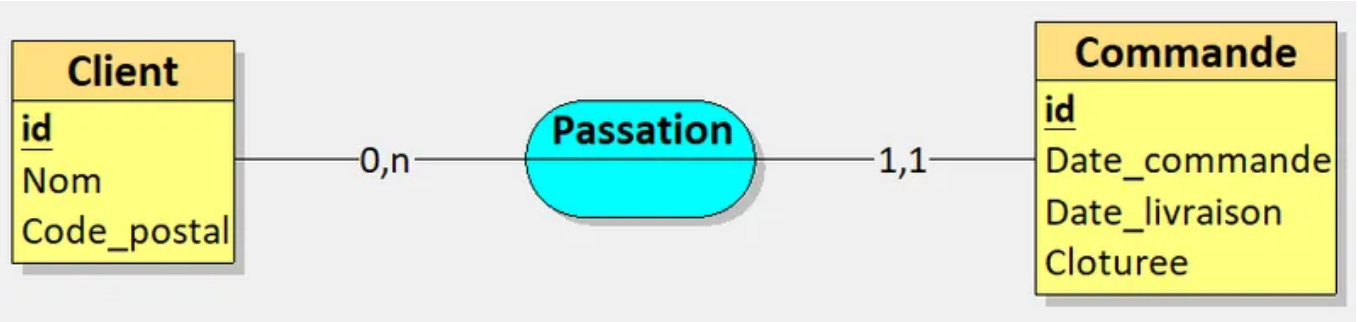
À la fin de cette leçon, tu dois être capable de :

- Expliquer à quoi servent les jointures en SQL
- Identifier les différents types de jointures
- Utiliser correctement les **deux notations** (implicite et explicite)
- Écrire des requêtes SQL avec des jointures pour combiner les données de plusieurs tables

📌 Pourquoi utiliser une jointure ?

Une **jointure** permet de **lier plusieurs tables** entre elles pour **croiser les informations**. Dans une base de données bien structurée (normalisée), les données sont réparties dans plusieurs tables. Par exemple :

- Une table `Clients`
- Une table `Commandes`



Si tu veux afficher le nom d'un client et les produits qu'il a commandés, tu devras **lier (joindre)** les deux tables à l'aide d'un champ commun, comme `id_client`. Une jointure te permettra d'afficher des données qui proviennent de 2 (ou plusieurs) tables et qui sont reliées entre elles.

🧠 Les notions essentielles à retenir

1. Une **jointure** combine les lignes de plusieurs tables.
2. Elle s'appuie toujours sur un ou plusieurs champs **communs** (souvent des clés primaires et étrangères).
3. Il existe **plusieurs types** de jointures : `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `FULL JOIN`.
4. Tu peux utiliser deux **syntaxes différentes** pour écrire une jointure : **implicite** et **explicite**.
5. L'`INNER JOIN` est le type le plus utilisé : il ne garde que les lignes **communes** aux deux tables.

✍️ Deux notations possibles

1. Notation implicite (ancienne, encore utilisée)

```sql
SELECT *
FROM Clients, Commandes
WHERE Clients.id = Commandes.id_client;
```

➡️ Avantage : simple à écrire, mais peu lisible pour des jointures complexes. ➡️ Inconvénient : facile d'oublier une condition de jointure, ce qui donne un produit cartésien (toutes les combinaisons possibles 😱).

2. Notation explicite (moderne, plus claire)

```sql
SELECT *
FROM Clients
JOIN Commandes ON Clients.id = Commandes.id_client;
```

➡️ C'est la méthode recommandée dans les projets professionnels.

🔗 Types de jointures

1. `INNER JOIN` – la plus courante

Elle **ne garde que les lignes correspondantes** dans les deux tables.

```sql
SELECT Clients.nom, Commandes.date
FROM Clients
INNER JOIN Commandes ON Clients.id = Commandes.id_client;
```

💡 Si un client **n'a pas encore commandé**, il **n'apparaîtra pas**.

2. `LEFT JOIN` – jointure externe gauche

Elle garde **tous les enregistrements de la table de gauche**, même si aucune correspondance dans la table de droite.

```sql
SELECT Clients.nom, Commandes.date
FROM Clients
LEFT JOIN Commandes ON Clients.id = Commandes.id_client;
```

💡 Utile pour lister **tous les clients**, même ceux **sans commande** (les dates seront `NULL`).

3. `RIGHT JOIN` – jointure externe droite

Inverse de `LEFT JOIN` : garde tous les éléments de la **table de droite**, même sans correspondance à gauche.

```sql
SELECT Clients.nom, Commandes.date
FROM Clients
RIGHT JOIN Commandes ON Clients.id = Commandes.id_client;
```

💡 Rarement utilisée. Peut être remplacée par un `LEFT JOIN` en inversant les tables.

4. `FULL OUTER JOIN` – toutes les correspondances

Elle retourne **tous les enregistrements** de chaque table, avec des `NULL` là où il n'y a pas de correspondance.

```sql
SELECT Clients.nom, Commandes.date
FROM Clients
FULL OUTER JOIN Commandes ON Clients.id = Commandes.id_client;
```

💡 Supportée uniquement dans certains SGBD (PostgreSQL, SQL Server... pas MySQL sans bidouille).

✅ Exemples concrets

Tables de départ

Table Clients

id	nom
1	Alice
2	Bob
3	Charlie

Table Commandes

id	id_client	date
1	1	2024-01-01
2	1	2024-01-05
3	2	2024-02-10

INNER JOIN

```sql
SELECT Clients.nom, Commandes.date
FROM Clients
JOIN Commandes ON Clients.id = Commandes.id_client;
```

Résultat :

nom	date
Alice	2024-01-01
Alice	2024-01-05
Bob	2024-02-10

Charlie n'apparaît pas car il n'a pas commandé.

LEFT JOIN

```sql
SELECT Clients.nom, Commandes.date
FROM Clients
LEFT JOIN Commandes ON Clients.id = Commandes.id_client;
```

Résultat :

nom	date
Alice	2024-01-01
Alice	2024-01-05
Bob	2024-02-10
Charlie	NULL

Charlie apparaît, mais n'a pas de date.

💡 Bonnes pratiques

- Privilégie toujours la **notation explicite**.
- Vérifie bien tes conditions de jointure (`ON ... = ...`) pour **éviter les doublons ou les croisements absurdes**.
- Utilise des **alias** pour raccourcir les noms :

```sql
SELECT c.nom, cmd.date
FROM Clients c
JOIN Commandes cmd ON c.id = cmd.id_client;
```

🧠 À retenir

- Les jointures sont **essentielles** pour relier les données.
- `INNER JOIN` pour les correspondances strictes.
- `LEFT JOIN` pour tout garder à gauche, même sans correspondance.
- Toujours préférer la **notation explicite**.
- Penser à bien identifier les **clés primaires** et **clés étrangères** entre les tables.

Souhaites-tu que je t'écrive une fiche pour l'enseignant (objectifs, déroulement de l'activité, pièges classiques, etc.) ?