Les jointures

En SQL (Structured Query Language), une **jointure** permet de combiner des données provenant de plusieurs tables en se basant sur une relation entre les colonnes de ces tables. Voici une présentation des différents types de jointures, illustrés avec des exemples.

1. INNER JOIN

- **Description** : Retourne les lignes où il y a une correspondance entre les deux tables.
- **Utilisation** : Pour obtenir uniquement les données partagées entre deux tables.

Exemple :

```
SELECT A.nom, B.salaire
FROM Employes A
INNER JOIN Salaires B ON A.id = B.employe_id;
```

Explication :

- On obtient les employés qui ont un salaire enregistré dans la table `Salaires`.

2. LEFT JOIN (ou LEFT OUTER JOIN)

- **Description** : Retourne toutes les lignes de la table de gauche, même si aucune correspondance n'est trouvée dans la table de droite. Les valeurs non correspondantes sont remplies par `NULL`.
- **Utilisation** : Pour lister toutes les données d'une table et les données associées, si elles existent.

Exemple :

```
SELECT A.nom, B.salaire
FROM Employes A
LEFT JOIN Salaires B ON A.id = B.employe_id;
```

Explication :

- On obtient tous les employés, avec leur salaire s'il est défini, ou `NULL` sinon.

3. RIGHT JOIN (ou RIGHT OUTER JOIN)

- **Description** : Retourne toutes les lignes de la table de droite, même si aucune correspondance n'est trouvée dans la table de gauche.
- **Utilisation** : Similaire à LEFT JOIN, mais pour la table de droite.

Exemple :

```
SELECT A.nom, B.salaire
FROM Employes A
RIGHT JOIN Salaires B ON A.id = B.employe_id;
```

Explication :

- On obtient tous les salaires, avec les noms des employés associés s'ils existent, ou `NULL` sinon.

4. FULL JOIN (ou FULL OUTER JOIN)

- **Description** : Retourne toutes les lignes des deux tables, avec les correspondances quand elles existent, sinon remplit par `NULL`.
- **Utilisation** : Pour combiner complètement les données de deux tables.

Exemple :

```
SELECT A.nom, B.salaire
FROM Employes A
FULL JOIN Salaires B ON A.id = B.employe_id;
```

Explication :

- On obtient tous les employés et tous les salaires, qu'il y ait une correspondance ou non.

5. CROSS JOIN

- **Description** : Retourne le produit cartésien des deux tables (toutes les combinaisons possibles des lignes).
- **Utilisation** : Rarement utilisé, mais utile pour générer toutes les combinaisons.

Exemple :

```
SELECT A.nom, B.salaire
FROM Employes A
CROSS JOIN Salaires B;
```

Explication :

- Chaque employé est associé à chaque salaire, sans condition de correspondance.

6. SELF JOIN

- **Description** : Une jointure d'une table avec elle-même.
- **Utilisation** : Pour comparer des lignes dans une même table.

Exemple :

```
SELECT A.nom AS Employe1, B.nom AS Employe2
FROM Employes A
INNER JOIN Employes B ON A.manager_id = B.id;
```

Explication :

- On obtient la liste des employés avec leur manager.

Schéma pour comprendre les jointures

Voici un schéma simplifié montrant le fonctionnement des jointures (A = table gauche, B = table droite) :

Type de jointure	Description
INNER JOIN	Intersection des tables (A ∩ B).
LEFT JOIN	Toutes les lignes de A, avec les correspondances dans B.
RIGHT JOIN	Toutes les lignes de B, avec les correspondances dans A.
FULL JOIN	Union complète des deux tables (A ∪ B).
CROSS JOIN	Produit cartésien des tables.

TP pratique sur les jointures en MySQL.

Ce TP te permettra de comprendre comment utiliser les différentes types de jointures dans des scénarios réels.

Création des Tables :

Crée et insère les données dans les tables.

```
-- Création de la table Employes
CREATE TABLE Employes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    salaire DECIMAL(10, 2),
    dept_id INT
);

-- Insertion des données dans Employes
INSERT INTO Employes (nom, salaire, dept_id) VALUES
('Alice', 4000, 1),
('Bob', 3000, 2),
('Charlie', 2500, NULL),
('David', 3500, 3);

-- Création de la table Departements
CREATE TABLE Departements (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL
);

-- Insertion des données dans Departements
INSERT INTO Departements (nom) VALUES
('Informatique'),
('RH'),
('Marketing'),
('Finance');
```

Exercice 1 : INNER JOIN

- **Question** : Liste tous les employés avec le nom de leur département.
- **Requête** :

```
SELECT Employes.nom AS Employe, Employes.salaire, Departements.nom AS Departement
FROM Employes
INNER JOIN Departements ON Employes.dept_id = Departements.id;
```

Exercice 2 : LEFT JOIN

- **Question** : Liste tous les employés, même ceux sans département.
- **Requête** :

```
SELECT Employes.nom AS Employe, Employes.salaire, Departements.nom AS Departement
FROM Employes
LEFT JOIN Departements ON Employes.dept_id = Departements.id;
```

Exercice 3 : RIGHT JOIN

- **Question** : Liste tous les départements, même ceux sans employés.
- **Requête** :

```
SELECT Employes.nom AS Employe, Departements.nom AS Departement
```

```
FROM Employes
RIGHT JOIN Departements ON Employes.dept_id = Departements.id;
```

Exercice 4 : FULL OUTER JOIN (Simulé)

- **Question** : Liste tous les employés et tous les départements, qu'ils soient liés ou non.
- **Requête** (MySQL ne supporte pas FULL OUTER JOIN nativement, mais on peut le simuler avec une UNION) :

```
SELECT Employes.nom AS Employe, Departements.nom AS Departement
FROM Employes
LEFT JOIN Departements ON Employes.dept_id = Departements.id
UNION
SELECT Employes.nom AS Employe, Departements.nom AS Departement
FROM Employes
RIGHT JOIN Departements ON Employes.dept_id = Departements.id;
```

Exercice 5 : Employés sans Département

- **Question** : Trouve tous les employés qui n'ont pas de département.
- **Requête** :

```
SELECT Employes.nom AS Employe, Employes.salaire
FROM Employes
WHERE dept_id IS NULL;
```

Exercice 6 : Nombre d'Employés par Département

- **Question** : Compte le nombre d'employés dans chaque département.
- **Requête** :

```
SELECT Departements.nom AS Departement, COUNT(Employes.id) AS NombreEmployes
FROM Departements
LEFT JOIN Employes ON Departements.id = Employes.dept_id
GROUP BY Departements.nom;
```