LES JOINTURES

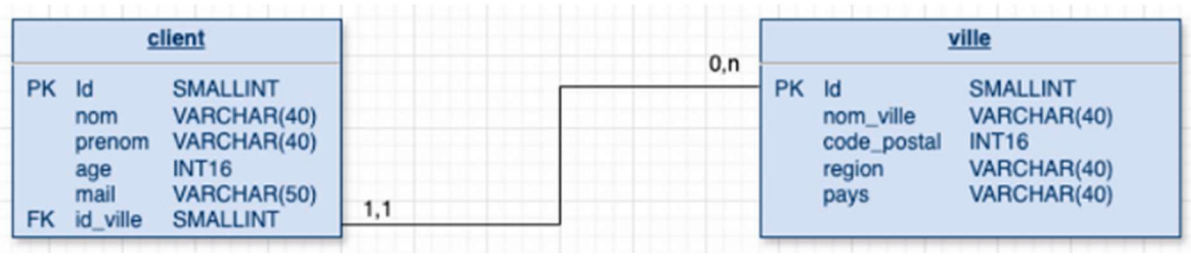
C'est quoi une jointure ?

Une requête avec jointure permet de relier plusieurs tables dans une base de données en utilisant une relation entre leurs colonnes (clé primaire / clé étrangère).

Elles sont utilisées pour combiner des données provenant de plusieurs tables. Les bases de données relationnelles stockent les données en plusieurs tables pour éviter la redondance.

Les jointures permettent de rassembler ces données réparties dans différentes tables lorsqu'on en a besoin.

Prenons le MCD ci-dessous :



Si nous souhaitons lister le nom d'un client avec le nom de la ville lui correspondant il faudra faire une jointure.

```
SELECT client.nom, ville.nom_ville
FROM client
INNER JOIN ville ON client.id_ville = ville.id ;
```

Cette requête sélectionne les colonnes nom de la table client ainsi que les nom_ville de la table ville.

Le INNER JOIN retourne les lignes ou il existe une correspondance entre les deux tables ville et client.

ON client.id_ville = ville.id permet de relier les deux tables via la clé étrangère id_ville de la table client et la clé primaire id de la table ville.

```
SELECT client.nom, ville.nom_ville
FROM client
INNER JOIN ville ON client.id_ville = ville.id
WHERE ville.nom_ville = 'Rodez' ;
```

Cette requête sélectionne les colonnes nom de la table client et nom_ville de la table ville.

Elle retourne les lignes où il existe une correspondance entre les deux tables grâce au INNER JOIN (jointure interne).

ON client.id_ville = ville.id permet de relier les deux tables via la clé étrangère id_ville de la table client et la clé primaire id de la table ville.

Et on ne veut que les noms de ceux qui habitent à Rodez.

IL EXISTE D'AUTRES TYPES DE JOINTURES QUI PEUVENT ETRE UTILES DANS CERTAINS CAS : JOIN, LEFT JOIN; RIGHT JOIN; FULL JOIN

1. INNER JOIN (ou simplement JOIN)

Comme nous l'avons vue renvoie uniquement les lignes présentes dans les deux tables.

Exemple :

```
SELECT client.nom, ville.nom_ville
FROM client
INNER JOIN ville ON client.id_ville = ville.id
```

Résultat : Affiche les clients qui ont une ville associée (il est possible de mettre une règle WHERE comme montré ci-dessus). Les clients sans ville renseignée (ou avec une ville qui n'existe pas dans la table ville) ne seront pas affichés.

2. LEFT JOIN (OU LEFT OUTER JOIN)

LEFT JOIN renvoie toutes les lignes de la table de gauche, même si aucune correspondance n'existe dans la table de droite. Les valeurs manquantes sont remplacées par NULL.

Exemple :

```
SELECT client.nom, ville.nom_ville
FROM client
LEFT JOIN ville ON client.id_ville = ville.id
```

Résultat : Affiche tous les clients, même ceux sans correspondance dans la table ville. S'il n'y a pas de ville correspondante, nom_ville sera NULL.

Dupont	**Paris**
Kelly	**Miami**
Emilio	**NULL**

Utile pour repérer les clients sans ville renseignée.

3. RIGHT JOIN (OU RIGHT OUTER JOIN)

Inverse du LEFT JOIN : renvoie toutes les lignes de la table de droite, même sans correspondance dans la table de gauche.

Exemple :

```
SELECT client.nom, ville.nom_ville
FROM client
RIGHT JOIN ville ON client.id_ville = ville.id ;
```

Résultat : Affiche toutes les villes, même celles sans client associé. Les clients inexistants pour une ville donneront NULL dans la colonne nom.

Dupont	**Paris**
Martin	**Marseille**
NULL	**Lyon**

4. FULL JOIN (OU FULL OUTER JOIN)

Renvoie toutes les lignes des deux tables, qu'il y ait correspondance ou non.

Exemple :

```
SELECT client.nom, ville.nom_ville
FROM client
FULL JOIN ville ON client.id_ville = ville.id;
```

Combine les effets du LEFT JOIN et du RIGHT JOIN :

- **Tous les clients, qu'ils aient une ville ou non**
- **Toutes les villes, qu'elles aient des clients ou non**

⚠ Tous les SGBD ne supportent pas FULL JOIN (par ex. MySQL avant la version 8 ne le gérait pas directement).