Enseignant :	Louis RAYNAL	**Établissement :**	ICES
Contact :	l-raynal@ices.fr	**Année :**	2025-2026
Public :	L3 Maths		
Ressources :	https://github.com/LouisRaynal/coursSQL		

Cours-TP 4 : Jointures entre tables

Nous avons brièvement abordé dans le cours-TP précédent qu'il est possible de spécifier plusieurs tables dans la clause `FROM` d'une requête de sélection. Il est alors nécessaire d'utiliser un mécanisme de **jointure** afin de réunir les informations des différentes tables. Nous allons maintenant approfondir ce sujet qui est sans aucun doute le plus important pour la maîtrise du langage SQL.

1. Jointures entre tables

Revenons à la première base de données que nous avons créée : `banque`, stockée dans le fichier `banque.db`. Elle contient trois tables avec les informations suivantes :

`clients`

id_client	nom	prénom
1	Blanc	Clémence
2	Dumont	Charles
3	Blake	George

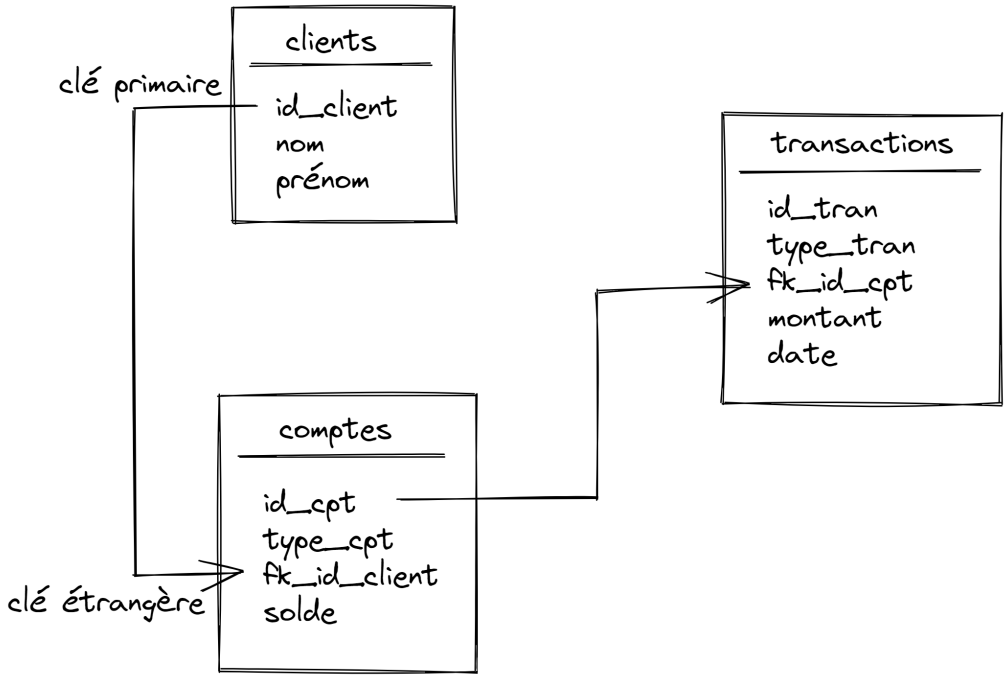
`comptes`

id_cpt	type_cpt	fk_id_client	solde
101	épargne	1	500.00
102	courant	1	250.00
103	courant	2	300.00
104	épargne	3	0.00
105	courant	3	1000.00

`transactions`

id_tran	type_tran	fk_id_cpt	montant	date
501	débit	102	55.00	2022-01-02
502	crédit	105	10000.00	2022-02-05
503	débit	105	9000.00	2022-02-06
504	débit	103	200.00	2022-02-08
505	crédit	101	500.00	2022-03-09

Pour cette base de données, le schéma relationnel est le suivant :



Maintenant que nous avons la structure de cette base de données en tête, nous pouvons essayer de joindre plusieurs tables ensemble, afin d'obtenir par exemple les informations des tables `clients` et `comptes` au sein d'une nouvelle table. Cela s'appelle faire une **jointure** entre des tables.

Pour faire une jointure on utilise l'opérateur `JOIN`, qui se décline sous différentes variantes (`INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN` et `OUTER JOIN`).

1.1. Jointure interne : `INNER JOIN`

Une **jointure interne** (`INNER JOIN`), permet de fusionner deux tables, via des **valeurs communes** qui existent entre ces tables. C'est là qu'interviennent **les valeurs des clés primaires et étrangères** ! En effet, les valeurs communes entre une clé primaire et une clé étrangère permettent de "raccorder" les tables entre elles pour en obtenir une nouvelle.

Pour faire une jointure interne entre une table `table_1` et une table `table_2` via **l'égalité des valeurs** d'une clé primaire de `table_1` et d'une clé étrangère de `table_2`, la syntaxe pour la clause `FROM` est la suivante :

```
...
FROM table_1
INNER JOIN table_2
    ON table_1.clé_primaire = table_2.clé_étrangère
...
```

Pratique

Sur la base de données `banque`, nous voulons par exemple fusionner les tables `comptes` et `clients`, via l'égalité des valeurs de leur colonne respective `fk_id_client` et `id_client`.
La requête permettant de faire cette jointure est la suivante :

Code 1 – Exemple de jointure interne entre les tables `comptes` et `clients`

```sql
SELECT *
FROM comptes
INNER JOIN clients
    ON comptes.fk_id_client = clients.id_client
;
```

On obtient alors la table fusionnée ci-dessous qui contient les colonnes des deux tables : d'abord les colonnes de `comptes` puis celles de `clients`. (Il s'agit de l'ordre d'apparition des tables dans la requête).

id_cpt	type_cpt	fk_id_client	solde	id_client	nom	prénom
101	épargne	1	500.00	1	Blanc	Clémence
102	courant	1	250.00	1	Blanc	Clémence
103	courant	2	300.00	2	Dumont	Charles
104	épargne	3	0.00	3	Blake	George
105	courant	3	1000.00	3	Blake	George

On remarque que chaque ligne vérifie bien la condition

```sql
comptes.fk_id_client = clients.id_client
```

Lors d'une jointure interne (`INNER JOIN`), vous pouvez intervertir l'ordre d'appel des tables, cela ne changera que l'ordre des colonnes de la nouvelle table.

Pratique

Pour vous en assurer, utilisez la requête suivante :

Code 2 – Exemple de jointure interne entre les tables `clients` et `comptes`

```sql
SELECT *
FROM clients
INNER JOIN comptes
    ON clients.id_client = comptes.fk_id_client
;
```

Après la clause `SELECT` vous pouvez évidemment lister les colonnes que vous souhaitez.

Attention

La commande `INNER JOIN` **ne conserve que les lignes des tables qui vérifient la condition de jointure.** En pratique : les lignes avec une valeur de clé primaire égale à une valeur de clé étrangère.

Pratique

Pour observer ce point d'attention, ajoutez un nouveau client dans la table `clients`, via la requête

```
INSERT INTO clients (id_client, nom, prénom) VALUES (4, 'Doe', 'John');
```

Relancez ensuite l'une des requêtes avec jointure interne. Le client John Doe n'apparaît pas dans le résultat de la requête, car son identifiant client `4` n'est pas retrouvé dans la table `comptes`.

Pratique

Exercice : Rédigez une requête de sélection qui effectue une jointure interne entre les tables `comptes` et `transactions`. Grâce à cette requête, affichez les informations : identifiant de transaction, type de transaction, identifiant du compte concerné et type de compte.

On remarquera que le compte avec pour identifiant `104` n'apparaît pas dans le résultat de la requête, car il n'y a aucune transaction associée à ce compte.

Les jointures ne se limitent pas à deux tables. Il n'y a pas de limite au nombre de jointures que l'on peut faire, il suffit d'ajouter des utilisations de `JOIN`.
Voici un exemple de syntaxe à utiliser pour fusionner trois tables :

```
...
FROM table_1
INNER JOIN table_2
    ON table_1.clé_primaire = table_2.clé_étrangère
INNER JOIN table_3
    ON table_2.clé_primaire = table_3.clé_étrangère
...
```

Nous supposons que les tables `table_2` et `table_3` sont liées entre elles via une clé primaire et étrangère. Il aurait été possible que la relation soit plutôt entre les tables `table_1` et `table_3`. Il aurait alors fallu transformer la dernière ligne pour faire apparaître la condition liant ces deux tables.

Pratique

Si l'on veut fusionner les tables `clients`, `comptes` et `transactions`, il faut effectuer plusieurs jointures : une jointure entre les tables `clients` et `comptes` via les identifiants de client, puis une jointure entre les tables `comptes` et `transactions` via les identifiants de compte.

Code 3 – Exemple de jointures internes impliquant trois tables

```
1 SELECT *
2 FROM clients
3 INNER JOIN comptes
4     ON clients.id_client = comptes.fk_id_client
5 INNER JOIN transactions
6     ON comptes.id_cpt = transactions.fk_id_cpt
7 ;
```

Exercice : Modifiez la requête ci-dessus pour afficher les informations : nom, prénom du client, date, montant et type de transaction réalisée ainsi que le type de compte concerné.

Remarque

> L'ordre dans lequel sont faites les jointures internes importe peu. Il faut juste que chaque condition de jointure soit correcte.

1.2. Jointure à gauche : `LEFT JOIN`

Rappelons que le client John Doe n'est pas inclus dans le résultat des requêtes précédentes car son identifiant client n'est pas présent dans la table `comptes` (il n'a pas de comptes), et que `INNER JOIN` conserve uniquement les lignes avec des valeurs de clé primaire et étrangère égales (ici l'identifiant client entre les tables `clients` et `comptes`).

Si l'on souhaite garder toutes les lignes d'une première table, bien que les valeurs de clé ne correspondent pas avec la deuxième table, nous pouvons utiliser `LEFT JOIN`. Cela permet de faire une **jointure (externe) à gauche**.

La syntaxe d'utilisation de `LEFT JOIN` est très similaire à celle d'`INNER JOIN` :

```
...
FROM table_1
LEFT JOIN table_2
    ON table_1.clé_primaire = table_2.clé_étrangère
...
```

Avec cette syntaxe, **toutes les lignes de `table_1` seront conservées** :

- les lignes de `table_1` vérifiant la condition de jointure avec les lignes de `table_2` seront correctement associées, comme avec un `INNER JOIN` ;
- les lignes de `table_1` sans correspondance avec `table_2` recevront des valeurs vides (`NULL`) pour les colonnes de `table_2`.

D'une manière générale, toutes les lignes de la table se trouvant avant (**à gauche**) le `LEFT JOIN` seront conservées, même s'il n'y a pas de lignes correspondantes dans la table après le `LEFT JOIN`.

Pratique

> Voici le code pour faire une jointure à gauche entre les tables `clients` et `comptes`
>
> Code 4 – Exemple de jointure à gauche
>
> ```
> 1 SELECT *
> 2 FROM clients
> 3 LEFT JOIN comptes
> 4     ON clients.id_client = comptes.fk_id_client
> 5 ;
> ```

Le résultat de cette requête est affiché ci-dessous. On constate que John Doe apparaît bien dans le résultat, et qu'il n'a pas de comptes dans cette banque.

id_client	nom	prénom	id_cpt	type_cpt	fk_id_client	solde
1	Blanc	Clémence	102	courant	1	250.00
1	Blanc	Clémence	103	épargne	1	500.00
2	Dumont	Charles	103	courant	2	300.00
3	Blake	George	105	courant	3	1000.00
3	Blake	George	104	épargne	3	0.00
4	Doe	John	NULL	NULL	NULL	NULL

Exercices :

Relancez la requête précédente en intervertissant les tables `clients` et `comptes`. Vous verrez que John Doe n'apparaît plus dans le résultat. En effet, c'est la table `comptes` qui est maintenant la table **à gauche** de `LEFT JOIN` et pour qui toutes les lignes seront conservées.

Rédigez une requête pour faire une jointure à gauche entre les tables `comptes` et `transactions`. Vous devriez avoir le compte dont l'identifiant est `104` qui est associé à des valeurs `NULL`, car il n'y a pas de transactions retrouvées pour ce compte.

Rédigez une requête de sélection avec deux jointures qui permettent de conserver toutes les lignes des tables `clients` et `comptes` tout en affichant les lignes de `transactions` correspondantes.

Attention

Mélanger un `LEFT JOIN` avec un `INNER JOIN` dans une même requête résultera toujours en une victoire du `INNER JOIN`, car une valeur de clé `NULL` n'a aucune correspondance dans une autre table.

Remarque

Dans la plupart des variantes de SQL, il existe les opérateurs `RIGHT JOIN` et `OUTER JOIN`. Ils ne sont pas acceptés par SQLite.

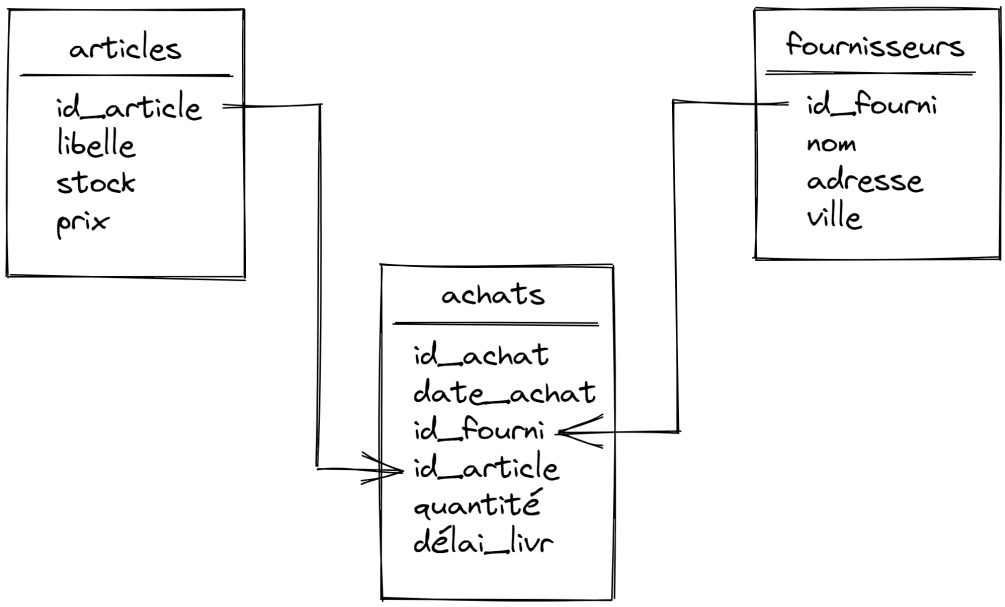
- `RIGHT JOIN` permet de conserver toutes les lignes de la table à sa droite. Cela équivaut à utiliser `LEFT JOIN` mais en intervertissant l'ordre d'appel des tables. En pratique il vaut mieux utiliser uniquement la jointure à gauche.
- `OUTER JOIN` permet de conserver toutes les lignes de la table à sa gauche et à sa droite.

Notez enfin qu'une même table peut être appelée deux fois pour faire une jointure avec elle-même.

2. Exercices

Pour ces exercices, vous utiliserez la base de données `commerce` qui contient les données d'achats (table `achats`) de certains articles (table `articles`) vendus par certains fournisseurs (table `fournisseurs`).

Le schéma relationnel de cette base est représenté ci-dessous.



1. Téléchargez le fichier de base de données `commerce.db` puis stockez le sur votre bureau, et connectez-vous y depuis SQLiteStudio.
2. Familiarisez vous avec le contenu de chaque table grâce à trois requêtes de sélection.
3. Affichez les identifiants et noms des articles dont le stock est inférieur à 10.
4. Affichez les lignes d'`articles` dont le prix est compris entre 100 et 300.
5. Affichez les fournisseurs dont on ne connaît pas l'adresse.
6. Affichez les fournisseurs dont le nom commence par "STE" et se termine pas "C".
7. Affichez les noms et adresses des fournisseurs qui proposent des délais de livraison supérieurs à 20 jours.
8. Affichez pour chaque article la valeur de son stock, en se limitant aux valeurs de stock supérieures à 1000.
9. Affichez les identifiants, noms et stocks des articles, triés par ordre décroissant des stocks.
10. Affichez les achats effectués en juillet 2017.
11. Listez les noms des articles pour lesquels le délai de livraison a été de 10 jours ou moins.
12. Affichez les 10 articles avec les délais de livraison les plus courts.
13. Affichez les dates d'achats avec les noms des articles achetés et les fournisseurs correspondants.
14. Affichez les fournisseurs qui n'ont fourni aucun des articles commandés.
15. Affichez les fournisseurs qui ont fourni les deux articles avec pour identifiant `100` et `106`, ainsi que leurs noms et prix.
16. Affichez toutes les colonnes d'`achats` ainsi que le prix total de la commande.
17. Affichez les noms et identifiants des articles ayant été achetés chez plusieurs fournisseurs. Affichez les lignes distinctes. (Rappel : une table peut être jointe avec elle-même).