SQL Avancé
Les jointures

Les jointures permettent de relier les tables entre elles. Ils existent plusieurs types de jointures la plus courante étant l'équi-jointure.

1. L'équi-jointure :

L'équi-jointure consiste à associer deux tables dont les valeurs des colonnes associées sont égales. Il est possible de réaliser cette jointure en utilisant l'opérateur INNER JOIN mais aussi avec des requêtes imbriquées.

1.1. Opérateur INNER JOIN :

Syntaxe :

```
SELECT [DISTINCT] <liste des attributs à projeter>
FROM <table>
[INNER] JOIN <table de jointure > ON <critère de jointure>;
-- le mot clé INNER est optionnel
```

CLIENTS (idClient, nomClient, prenomClient, villeClient)

C1	Terrieur	Alain	Montpellier
C2	Terrieur	Alex	Nimes
C3	Bricot	Judas	Montpellier
C4	Nanas	Judas	Montpellier

ACHATS (idAchat, montantAchat, idClient#)

A1	100	C1
A2	150	C1
A3	90	C1
A4	110	C2
A5	1000	C2
A6	600	C4
A7	1200	NULL

Exemple : Afficher le numéro et le nom des clients qui ont effectué au moins un achat.

Résultat attendu :

```
idClient    nomClient
----------- ----------
C1          Terrieur
C2          Terrieur
C4          Nanas
```

```
SELECT DISTINCT c.idClient, nomClient
FROM Clients c
JOIN Achats a ON c.idClient = a.idClient;
```

Cette requête produit une nouvelle table contenant les colonnes des deux tables jointes qui respectent la condition de jointure :

```
c.idClient  nomClient  prenomClient  villeClient  idAchat  montantAchat  a.idClient
----------- ---------- ------------- ------------ -------- ------------- -----------
C1          Terrieur   Alain         Montpellier    A1       100         C1
C1          Terrieur   Alain         Montpellier    A2       150         C1
C1          Terrieur   Alain         Montpellier    A3        90         C1
C2          Terrieur   Alex          Nimes          A4       110         C3
C2          Terrieur   Alex          Nimes          A5      1000         C3
C4          Nanas      Judas         Montpellier    A6       600         C4
```

Avant la norme SQL2 de 1992, la jointure s'écrivait ainsi :

```
SELECT DISTINCT c.idClient, nomClient
FROM Clients c, Achats a
WHERE a.idClient = c.idClient;
```

Depuis la norme SQL2, il est déconseillé d'utiliser cette notation car elle est moins claire lorsqu'il y a beaucoup de jointures et de conditions de sélection. De plus, si on oublie la condition de jointure dans le WHERE, la requête effectuera un produit cartésien au lieu d'une jointure.

1.2. Requêtes imbriquées avec opérateur IN :

Il est également possible de réaliser une jointure avec une requête imbriquée.

```
SELECT idClient, nomClient
FROM Clients
WHERE idClient IN (SELECT idClient
                   FROM Achats);
```

Contrairement à l'opérateur JOIN, cette requête ne produit pas une nouvelle table contenant l'association des deux tables reliées. Cette requête s'exécute en deux temps. La sous-requête est exécutée en premier de façon indépendante et retourne une liste de valeurs. Puis la requête principale est exécutée en remplaçant la condition du WHERE par cette liste de valeurs.

```
SELECT idClient, nomClient
FROM Clients
WHERE idClient IN ('C1', 'C1', 'C1', 'C2',  'C2', 'C4') ;
```

La forme imbriquée n'est pas la plus performante pour effectuer une jointure mais elle va être indispensable quand l'utilisation d'un DISTINCT dans un SELECT donne un mauvais résultat.

Exemple : Afficher le nom des clients qui ont effectué au moins un achat.

~~SELECT DISTINCT nomClient~~
~~FROM Clients c~~
~~JOIN Achats a ON c.idClient = a.idClient;~~

Le résultat obtenu est faux car il manque un client :

```
nomClient
----------
Terrieur
Nanas
```

```
SELECT nomClient
FROM Clients
WHERE idClient IN (SELECT idClient
                   FROM Achats);
```

Résultat obtenu :

```
nomClient
----------
Terrieur
Terrieur
Nanas
```

Attention à l'utilisation du DISTINCT :
On remarque dans la requête précédente que l'utilisation du DISTINCT donne un mauvais résultat car il est utilisé sur un attribut (ici le nom du client) dont les valeurs ne sont pas uniques. L'ajout de la clé primaire idClient dans le SELECT permettrait d'éviter ce problème. Il ne faut donc pas utiliser l'opérateur DISTINCT si la clé primaire n'est pas présente dans le SELECT. Mais si on ne souhaite pas afficher la clé primaire, la seule solution est d'utiliser la requête imbriquée pour réaliser la jointure.

Il est à noter que l'utilisation du DISTINCT n'est pas systématique quand la clé primaire est présente dans le SELECT. Cela dépend de la relation entre les deux tables.
Dans notre exemple, un client peut réaliser plusieurs achats (relation un - plusieurs) mais un achat est réalisé par un seul client (plusieurs - un).

Relation un - plusieurs : ici il faut mettre un DISTINCT car dans la table produit par la jointure le client est dupliqué pour chaque achat effectué. Il y a donc des doublons sur les clients.

```
SELECT DISTINCT c.idClient, nomClient
FROM Clients c
JOIN Achats a ON c.idClient = a.idClient;
```

Relation plusieurs - un : le DISTINCT est inutile car dans la table produit par la jointure un achat n'apparait qu'une seule fois. Il n'y a donc pas de doublons sur les achats.

```
SELECT c.idAchat, montantAchat
FROM Achats a
JOIN Clients c ON c.idClient = a.idClient;
```

1.3. Opérateur EXISTS :

L'opérateur EXISTS permet également de réaliser une jointure. Il consiste à vérifier si une sous-requête retourne un résultat ou non. Si la sous-requête contient des lignes, la réponse est TRUE, sinon la réponse est FALSE.

```
SELECT idClient, nomClient
FROM Clients c
WHERE EXISTS (SELECT *
              FROM Achats a
              WHERE a.idClient = c.idClient);
```

Bien que la requête avec l'opérateur EXISTS ait la forme d'une requête imbriquée, son exécution n'est par la même. Contrairement à la requête imbriquée avec l'opérateur IN, la sous-requête n'est pas exécutée de façon indépendante à la requête principale. Ici la sous-requête est exécutée pour chaque ligne de la requête principale car la sous-requête est corrélée à la requête principale via la condition de jointure dans le WHERE de la sous-requête

2. JOINTURE EXTERNE

Ce type de jointure permet de d'extraire toutes le données d'une table même s'il n'y a pas d'association avec la table jointe.

Syntaxe :

```
SELECT <liste des attributs à projeter>
FROM <table>
LEFT|RIGHT [OUTER] JOIN <table de jointure > ON <critère de jointure>;
-- le mot clé OUTER est optionnel
```

Jointure externe gauche :

On souhaite afficher tous les clients de la table CLIENTS même s'ils n'ont effectué aucun achat.

```
SELECT *
FROM Clients c
LEFT OUTER JOIN Achats a ON c.idClient = a.idClient;
```

Résultat obtenu :

c.idClient	nomClient	prenomClient	villeClient	idAchat	montantAchat	a.idClient
C1	Terrieur	Alain	Montpellier	A1	100	C1
C1	Terrieur	Alain	Montpellier	A2	150	C1
C1	Terrieur	Alain	Montpellier	A3	90	C1
C2	Terrieur	Alex	Nimes	A4	110	C3
C2	Terrieur	Alex	Nimes	A5	1000	C3
C3	**Bricot**	**Judas**	**Montpellier**	**NULL**	**NULL**	**NULL**
C4	Nanas	Judas	Montpellier	A6	600	C4

Jointure externe droite :

On souhaite afficher tous les achats de la table ACHATS même s'ils n'ont été effectués par aucun client.

```
SELECT *
FROM Clients c
RIGHT OUTER JOIN Achats a ON c.idClient = a.idClient;
```

Résultat obtenu :

c.idClient	nomClient	prenomClient	villeClient	idAchat	montantAchat	a.idClient
C1	Terrieur	Alain	Montpellier	A1	100	C1
C1	Terrieur	Alain	Montpellier	A2	150	C1
C1	Terrieur	Alain	Montpellier	A3	90	C1
C2	Terrieur	Alex	Nimes	A4	110	C3
C2	Terrieur	Alex	Nimes	A5	1000	C3
C4	Nanas	Judas	Montpellier	A6	600	C4
NULL	**NULL**	**NULL**	**NULL**	**A7**	**1200**	**NULL**