

# Les jointures SQL

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. Généralités sur le modèle relationnel</b>	<b>3</b>
<b>III. Exercice : Appliquez la notion</b>	<b>4</b>
<b>IV. Les jointures</b>	<b>5</b>
<b>V. Exercice : Appliquez la notion</b>	<b>9</b>
<b>VI. Écrire une jointure</b>	<b>9</b>
<b>VII. Exercice : Appliquez la notion</b>	<b>13</b>
<b>VIII. Les alias de table</b>	<b>13</b>
<b>IX. Exercice : Appliquez la notion</b>	<b>15</b>
<b>X. Les clés de table</b>	<b>16</b>
<b>XI. Exercice : Appliquez la notion</b>	<b>18</b>
<b>XII. Essentiel</b>	<b>19</b>
<b>XIII. Auto-évaluation</b>	<b>19</b>
A. Exercice final .....	19
B. Exercice : Défi .....	21
<b>Solutions des exercices</b>	<b>22</b>

## I. Contexte

**Durée** : 1 h

**Environnement de travail** : MariaDB, DataGrip

**Pré-requis** : Connaître les principes d'une base de données relationnelle, connaître les requêtes simples

### Contexte

Un SGBD relationnel conserve ses données sous forme de tables. Il est très rare que toutes les données nécessaires à une requête ne soient que dans une seule table, c'est pourquoi il sera souvent nécessaire d'aller les chercher dans plusieurs tables séparées.

Il est bien évidemment possible d'effectuer une requête `SELECT` par table, puis d'effectuer un croisement des données, mais ce n'est ni pratique, ni performant.

La meilleure façon d'accéder aux données de plusieurs tables en même temps est d'interroger toutes ces tables au moyen de jointures.

## II. Généralités sur le modèle relationnel

### Objectifs

- Comprendre les avantages du modèle relationnel
- Éviter la redondance de données

### Mise en situation

Lors de la création d'une base de données, il faut éviter la redondance de données, c'est-à-dire le fait qu'une donnée soit répétée dans plusieurs tables. Pour cela, il faut séparer les informations sur plusieurs tables et les lier entre elles.

### Méthode Découper son schéma

Imaginons que nous devons créer une base de données gérant la liste des animaux pour une garderie. On a besoin de savoir, pour chaque animal, leur espèce, leur race, leur régime alimentaire, leur nom et le nom de leur propriétaire. Il serait possible de stocker toutes ces données dans une seule table :

Id	Espèce	Race	Régime	Nom	Propriétaire
1	Chien	Husky	Carnivore	Fantôme	Dupond
2	Chat	Maine Coon	Carnivore	Ficelle	César
3	Chien	Berger allemand	Carnivore	Mirza	Dupond
4	Lapin	Angora	Végétarien	Doudou	Fanny

Ce schéma pose plusieurs problèmes de redondance de données. Le premier est la colonne "Régime" : il existe un lien entre une espèce d'animal et son régime alimentaire. Ainsi, nous savons que tous les chiens sont des carnivores, ce n'est pas nécessaire de le rappeler à chaque ligne. Nous pouvons donc séparer cette information dans une table à part.

Il en est de même pour les propriétaires : Dupond possède deux animaux, mais son nom apparaît deux fois dans la table, ce qui cause une redondance.

Un meilleur schéma pourrait ressembler à celui-ci :

Espèces			Propriétaires			Animaux				
id	Nom	Regime	id	Nom		id	Especie	Race	Nom	Proprietaire
1	Chien	Carnivore	1	Dupond		1	1	Husky	Fantôme	
2	Chat	Carnivore	2	César		2	2	Maine Coon	Ficelle	
3	Lapin	Végétarien	3	Fanny		3	1	Berger allemand	Mirza	
						4	3	Angora	Doudou	

Cette fois-ci, les données sont réparties sur trois tables. La table "Animaux" contient à présent des numéros qui correspondent aux identifiants des lignes des autres tables : le numéro de la colonne "Espèce" correspond à la colonne ID de la table "Espèces", et la colonne "Propriétaire" correspond à la colonne ID de la table "Propriétaires".

Ce schéma est plus complexe, mais sera plus efficace et plus pratique à l'usage. Par exemple, si un propriétaire se plaint que son nom a été mal orthographié, il n'y aura qu'une ligne à changer dans la table des propriétaires, et non plusieurs lignes dans l'ancienne table des animaux.

#### Remarque

Une base de données relationnelle est constituée de multiples tables liées entre elles par des références d'une table à l'autre.

Dans une base bien conçue, une même donnée ne se trouve qu'à un seul endroit.

#### Syntaxe À retenir

- Un bon schéma relationnel doit éviter la redondance des données.
- Les données seront donc réparties sur plusieurs tables liées entre elles par des références.

#### Complément

[https://fr.wikipedia.org/wiki/Mod%C3%A8le\\_relationnel](https://fr.wikipedia.org/wiki/Mod%C3%A8le_relationnel)

## III. Exercice : Appliquez la notion

### Question 1

[solution n°1 p.23]

Un club de vacances organise différentes activités pour ses participants, qui doivent réserver une date pour chaque activité qu'ils souhaitent faire. Tout ce processus est informatisé et le club de vacances utilise la base de données suivante :

Id	NomActivite	LieuActivite	Date	NomParticipant	PrenomParticipant	TelephoneParticipant
1	Planche à voile	Plage	2030-07-06	Jean	Dupont	0123456789
2	Skateboard	Skatepark	2030-07-06	John	Doe	0147852369
3	Planche à voile	Plage	2030-07-07	Laure	Mondi	0213467958
4	VTT	Forêt	2030-07-08	Jean	Dupont	0123456789
5	VTT	Forêt	2030-07-08	Laure	Mondi	0213467958

Cette base de données comporte de la redondance : les informations des participants et des activités sont dupliquées. Dans un premier temps, externalisez les informations des participants dans une table à part, et faites les modifications nécessaires pour lier une réservation d'activité à son participant.

#### Indice :

Pour lier la réservation au participant, il faut rajouter une colonne contenant l'ID du participant à la réservation. Cela signifie qu'un participant doit également posséder son propre ID.

## Question 2

[solution n°2 p.23]

Les activités sont également redondantes : chaque activité se fait toujours dans le même lieu, mais cette information est dupliquée. Externalisez les informations liées aux activités (nom et lieu) dans une autre table.

## IV. Les jointures

### Objectifs

- Comprendre le concept de jointure
- Savoir utiliser les jointures pour récupérer les données de plusieurs tables

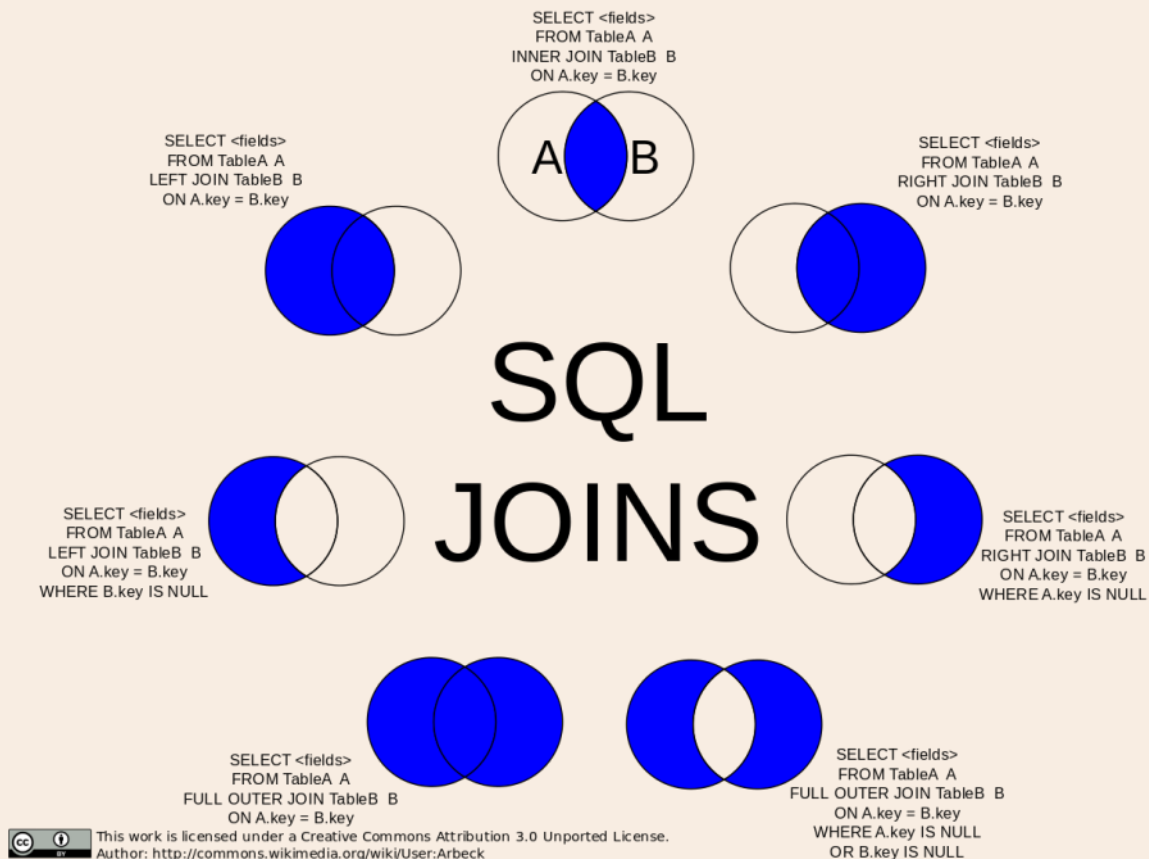
### Mise en situation

Puisqu'une base de données relationnelle est composée de plusieurs tables liées entre elles, il va falloir rassembler les données de ces diverses tables. Pour cela, il faut effectuer une **jointure** entre les tables au moment de la requête de lecture des données.

Définition	Les jointures
Une jointure est une opération de lecture de données entre deux tables permettant d'obtenir une table virtuelle contenant le croisement de ces données. Une jointure nécessite de préciser le lien entre deux tables.	
On peut regrouper ces jointures en 3 types essentiels :	
<ul style="list-style-type: none"><li>• Les jointures <b>internes</b>, qui permettent de croiser les données présentes dans deux tables. Seules les lignes ayant des données en commun ressortiront de la requête.</li><li>• Les jointures <b>gauches</b> (ou <b>droites</b>), qui permettent de compléter les données d'une table avec celles présentes dans une seconde. Toutes les lignes de la table gauche (ou droite) ressortiront de la requête, et complétées avec les données de l'autre table pour les lignes ayant des données en commun (ou des valeurs nulles sinon).</li><li>• Les jointures <b>externes</b>, qui permettent de rassembler les données de deux tables, y compris les données non communes. Toutes les lignes des deux tables ressortiront de la requête.</li></ul>	

Complément

Ces trois types essentiels peuvent être déclinés, selon les données que l'on souhaite récupérer :



Illustrons chaque cas avec un schéma simple : des utilisateurs appartenant à des groupes.

Users		
id	username	group_id
1	John	1
2	Laure	2
3	Robert	NULL

Groups	
id	name
1	Administrateurs
2	Modérateurs
3	Relecteurs

La **jointure interne** INNER de Users et de Groups retournera les données de tous les utilisateurs possédant un groupe et de tous les groupes ayant des utilisateurs.

id	username	group_id	id	name
1	John	1	1	Administrateurs
2	Laure	2	2	Modérateurs

La **jointure droite** RIGHT de Users et de Groups retournera les données de tous les groupes avec les utilisateurs associés, s'il y en a.

id	username	group_id	id	name
1	John	1	1	Administrateurs
2	Laure	2	2	Modérateurs
NULL	NULL	NULL	3	Relecteurs

La **jointure droite** `RIGHT` avec exclusion des données présentes dans `Users` de `Groups` ne retournera que les groupes n'ayant pas d'utilisateurs.

id	username	group_id	id	name
NULL	NULL	NULL	3	Relecteurs

La **jointure externe** `OUTER` avec exclusion des données présentes simultanément dans les deux tables retournera tous les groupes n'ayant pas d'utilisateurs et les utilisateurs n'ayant pas de groupes.

id	username	group_id	id	name
3	Robert	NULL	NULL	NULL
NULL	NULL	NULL	3	Relecteurs

La **jointure externe** `OUTER` des données présentes simultanément dans les deux tables retournera tous les groupes et tous les utilisateurs, en faisant un lien si possible.

id	username	group_id	id	name
1	John	1	1	Administrateurs
2	Laure	2	2	Modérateurs
3	Robert	NULL	NULL	NULL
NULL	NULL	NULL	3	Relecteurs

Les **deux jointures gauche** `LEFT` (avec et sans exclusion) suivent la même logique que les `RIGHT`, mais en se basant sur les données de `Users` plutôt que de `Groups`.

#### Remarque

Une jointure gauche de A vers B et une jointure droite de B vers A aboutissent au même résultat.

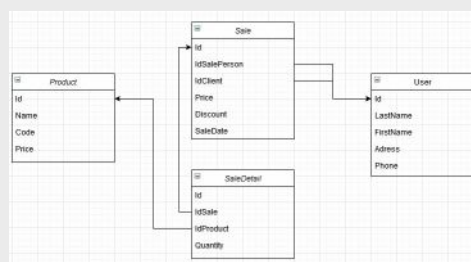
Dans la pratique, la jointure gauche sera pratiquement toujours employée.

#### Exemple Obtenir le détail d'une vente

Prenons l'exemple d'un schéma plus complet, qui nous servira d'exemple pour le reste du chapitre.

Un magasin a besoin de pouvoir conserver la liste des ventes effectuées. Il y a pour cela plusieurs informations à retenir : les informations du vendeur et de l'acheteur (nom, prénom...), celles du produit acheté (nom, prix de base...), le contenu de la commande (quels produits ont été achetés, et en quelle quantité) et les informations de la vente finale (prix, remise effectuée, date...).

Toutes ces informations sont stockées dans les tables suivantes :



Avant de faire des requêtes sur ce schéma, il est nécessaire de le comprendre. Les produits sont stockés dans la table `Product`. Les vendeurs et les acheteurs sont tous stockés dans la table `User`. Les ventes sont représentées par la table `Sale` : c'est ici que l'on détermine quel `User` est l'acheteur et lequel est le vendeur (avec `idSalePerson` et `idClient`). Enfin, la liste des produits achetés par vente sont dans la table `SaleDetail` : pour une vente et un produit, on y retrouve la quantité achetée.

Par exemple, si Jean Dupont achète deux chaises et une table à John Doe, le contenu des tables ressemblera à :

Sale					
Id	IdSalePerson	IdClient	Price	Discount	SaleDate
1	1	2	100	10	2032-07-08

SaleDetail			
Id	IdSale	IdProduct	Quantity
1	1	1	2
2	1	2	1

User				
Id	LastName	FirstName	Address	Phone
1	Doe	John	10 rue Danton	0123456789
2	Jean	Dupont	36 rue de la mer	0198765432

Product			
Id	Name	Code	Price
1	Chaise	01-CHAIR	10
2	Table de jardin	05-TABLE	80

Si on veut obtenir le détail des ventes, donc récupérer tous les produits liés à toutes les ventes, il va falloir joindre les données des tables SaleDetail et Product, en rassemblant sur la colonne qu'elles ont en commun l'ID de la commande (représenté par les flèches oranges). Nous allons donc partir de la table SaleDetail et faire une jointure gauche sur la table Product. Voici un exemple de résultat d'une jointure gauche entre ces deux tables, filtré sur une commande factice ayant pour ID 1 :

Jointure gauche de SaleDetail et de Product									
SaleDetail				Product					
Id	IdSale	IdProduct	Quantity	Id	Name	Code	Price		
1	1	2	4	2	Chaise	01-CHAIR	10		
2	1	7	1	7	Table de jardin	05-TABLE	80		
3	1	3	4	3	Coussins	ACCESSORY01	5		
4	1	2	2	2	Chaise	01-CHAIR	10		

### Remarque

Dans cet exemple, la table SaleDetail contient deux lignes de vente pour des chaises : les lignes numéro 1 et 4. Il n'y a qu'une seule ligne représentant des chaises dans la table Product, mais, dans la jointure, cette ligne sera dupliquée : notre client a acheté un lot de quatre chaises, et un autre lot de deux chaises.

Pour rappel, le résultat de la jointure est une table virtuelle. La ligne pour les chaises dans la table Product n'a pas été dupliquée, le contenu des données n'a pas été changé.

### Syntaxe À retenir

- Une jointure permet de rassembler les données de deux tables différentes. On jointe par rapport aux valeurs de deux colonnes, une par table.
- Le résultat de cette jointure est une table virtuelle.

### Complément

[https://fr.wikipedia.org/wiki/Jointure\\_\(informatique\)](https://fr.wikipedia.org/wiki/Jointure_(informatique))



## V. Exercice : Appliquez la notion

### Question 1

[solution n°3 p.23]

Une faculté utilise une base de données pour gérer ses étudiants et ses classes. Chaque étudiant est assigné à une classe, et à chaque classe est allouée une certaine quantité de matériel : des équerres, des rapporteurs, etc. Voici la structure de la base de données avec un exemple de données que l'on peut y trouver :

Etudiant				Classe			MatérielAlloue			Matériel		
Id	Nom	Prenom	IdClasse	Id	Nom	Spécialité	IdClasse	IdMatériel	Quantite	Id	Nom	QuantiteTotale
1	Dupont	Jean	1	1	M.A.T.	Mathématiques avancées	1	1	15	1	Equerre	50
2	Laure	Mondi	1	2	M.A.	Arts plastiques et modélisation 3D	1	3	10	2	Pinceau	25
3	John	Doe	2				2	2	20	3	Rapporteur	25
							2	3	10			

Sur quelles colonnes de quelles tables faut-il faire une jointure pour récupérer les informations de tous les étudiants et de la classe dans laquelle ils sont ?

#### Indice :

Pour trouver les colonnes qui vont nous aider à faire notre jointure, il faut trouver, dans deux tables différentes, une colonne qui désigne la même chose.

### Question 2

[solution n°4 p.23]

Sur le même schéma, comment récupérer la quantité de matériel nécessaire pour chaque classe, avec le nom du matériel associé ?

#### Indice :

Il va falloir une jointure sur plus de deux tables.

## VI. Écrire une jointure

### Objectifs

- Savoir écrire une jointure en SQL
- Récupérer des données réparties sur plusieurs tables en une requête

### Mise en situation

Pour récupérer des données dans une base de données, nous avons utilisé des requêtes sous la forme `SELECT ... FROM ... WHERE ...`. Si nos données sont réparties sur plusieurs tables, il va falloir faire une jointure. Pour cela, nous allons rajouter une étape dans notre `SELECT` : le `JOIN ... ON`.

#### Syntaxe Les jointures

Une jointure se fait via le mot-clé `JOIN`. Sans plus de précisions, il s'agira d'une jointure **interne**, mais il est possible de préciser le type de jointure :

- `INNER JOIN` (ou `JOIN`) : **interne**
- `LEFT JOIN` : **gauche**
- `RIGHT JOIN` : **droite**
- `OUTER JOIN` : **externe**

Il faut également préciser sur quel champ se fait la jointure. En effet, une table en référence une autre via un identifiant, qu'il faut mentionner dans la requête. Cela se fait au moyen du mot-clé `ON` suivi d'une ou plusieurs égalités. L'instruction complète devient donc :

```
1 SELECT
2     colonne1, colonne2
3 FROM
4     Table1
5     (INNER / LEFT / RIGHT / OUTER) JOIN Table2
6     ON idReference = id
```

### Attention

Dans MariaDB, les `OUTER JOIN` n'existent pas. On peut cependant les simuler avec des `UNION`. Pour l'exclusion, il faut utiliser :

```
1 SELECT
2     users.id, username, group_id, groups.id, name
3 FROM
4     users
5 LEFT JOIN
6     groups ON users.group_id = groups.id
7 WHERE
8     groups.id IS NULL
9 UNION
10 SELECT
11     users.id, username, group_id, groups.id, name
12 FROM
13     groups
14 LEFT JOIN
15     users ON users.group_id = groups.id
16 WHERE
17     users.id IS NULL
```

Et sans l'exclusion :

```
1 SELECT
2     users.id,
3     username,
4     group_id,
5     groups.id,
6     name
7 FROM
8     users
9 LEFT JOIN
10     groups ON users.group_id = groups.id
11 UNION
12 SELECT
13     users.id,
14     username,
15     group_id,
16     groups.id,
17     name
18 FROM
19     groups
20 LEFT JOIN
21     users ON users.group_id = groups.id
```

### Exemple

Reprenons notre base de données de produits. Pour obtenir le détail des lignes de ventes, tel que montré dans le tableau suivant :

SaleDetail				Product		
Id	IdSale	IdProduct	Quantity	Id	Name	Code
1	1	2	4	2	Chaise	01-CHAIR
2	1	7	1	7	Table de jardin	05-TABLE
3	1	3	4	3	Coussins	ACCESSORY01
4	1	2	2	2	Chaise	01-CHAIR

Il faut employer l'instruction suivante :

```

1 SELECT
2     *
3 FROM SaleDetail
4     INNER JOIN Product
5         ON Product.Id = SaleDetail.IdProduct

```

La relation entre les deux tables est définie par : `ON Product.Id = SaleDetail.IdProduct`. Cela signifie que les lignes de la table SaleDetails seront regroupées avec les lignes de la table Product dont l'ID correspond au champ IdProduct.

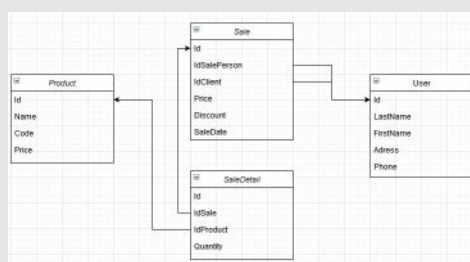
### Remarque Préfixer les champs

Dans notre jointure, nous avons préfixé les noms des colonnes du nom de la table. Ce n'est pas toujours obligatoire, mais c'est une bonne pratique. Préciser le nom de la table est nécessaire dans les cas où un nom de colonne est ambigu, c'est-à-dire qu'un même nom peut désigner plusieurs colonnes différentes. Dans notre cas, en écrivant simplement `ON Id = IdProduct`, la colonne Id pourrait désigner la colonne de la table Product ou celle de la colonne SaleDetail : il y a ambiguïté et le SGBD lèverait une erreur.

Pour le champ IdProduct, il n'y a pas d'ambiguïté, mais cela permet d'avoir une requête plus claire.

### Méthode Joindre plus de deux tables

Étant donné que le résultat d'une jointure est une table virtuelle, il est possible d'effectuer une nouvelle jointure avec une autre table. Ainsi, dans une même instruction `SELECT`, on va pouvoir enchaîner plusieurs `JOIN`. Reprenons nos commandes :



Pour le moment, nous ne récupérons que le détail de la commande, mais il est possible d'ajouter une jointure pour récupérer les informations de la vente, contenues dans la table Sale. Nous allons donc réaliser une première jointure entre Sale et SaleDetail, puis sur SaleDetail et Product, comme précédemment.

```

1 SELECT
2     *
3 FROM Sale
4     INNER JOIN SaleDetail
5         ON SaleDetail.IdSale = Sale.Id
6     INNER JOIN Product
7         ON Product.Id = SaleDetail.IdProduct

```

Le résultat final comportera les colonnes de ces trois tables :

Sale						SaleDetail				Product			
id	idSalePerson	idClient	Price	Discount	SaleDate	id	idSale	idProduct	Quantity	id	Name	Code	Price
1	2	5	160	10	2030-07-08	1	1	2	4	2	Chaise	01-CHAIR	10
1	2	5	160	10	2030-07-08	2	1	7	1	7	Table de jardin	05-TABLE	80
1	2	5	160	10	2030-07-08	3	1	3	4	3	Coussins	ACCESSORY01	5
1	2	5	160	10	2030-07-08	4	1	2	2	2	Chaise	01-CHAIR	10

Les flèches désignent les valeurs utilisées pour les jointures.

### Attention

Dans une jointure, pour toutes les lignes d'une table A, on associe toutes les lignes d'une table B qui satisfont la condition donnée dans le `ON`, ce qui peut provoquer des duplicatas. Comme on peut le voir dans cet exemple, plusieurs lignes de `Sale` et de `Product` sont dupliquées dans le résultat final : c'est quelque chose dont il faut tenir compte lors du traitement du résultat final.

### Complément Une autre manière de joindre

Il existe une autre syntaxe pour joindre plusieurs tables, mais qui est fortement déconseillée, car peu lisible. Toutefois, il est important de la connaître, car il est possible de la trouver dans du code déjà existant.

Cette syntaxe consiste à ne pas utiliser le mot-clé `JOIN`, mais à faire la jointure directement dans le `WHERE` d'un `SELECT`. L'exemple précédent devient donc :

```

1 SELECT
2     *
3 FROM Sale, SaleDetail, Product
4 WHERE
5     SaleDetail.IdSale = Sale.Id
6     AND Product.Id = SaleDetail.IdProduct

```

Le problème de cette syntaxe est qu'il devient difficile de distinguer ce qui est une jointure de ce qui est une simple condition dans la clause `WHERE`, c'est pourquoi elle a été abandonnée au profit du `JOIN`.

### Syntaxe À retenir

- L'instruction pour écrire une jointure est : `:(INNER ou LEFT ou RIGHT ou OUTER) JOIN NomDeTable ON Condition.`

### Complément

<https://mariadb.com/kb/en/join-syntax/>

## VII. Exercice : Appliquez la notion

### Question

[solution n°5 p.23]

En disposant des deux tables User et Sale, définies comme suit :

```
1 CREATE TABLE User
2 (
3     Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
4     LastName varchar(50) NOT NULL,
5     FirstName varchar(50) NOT NULL,
6     Adress varchar(255) NOT NULL,
7     Phone varchar(20) NOT NULL
8 )
9 ;
10 CREATE TABLE Sale
11 (
12     Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
13     IdSalePerson int NOT NULL,
14     IdClient int,
15     Price decimal(10,2) NOT NULL,
16     Discount decimal(10,2) NOT NULL,
17     SaleDate date DEFAULT NOW()
18 )
19 ;
```

Complétez la requête SELECT suivante afin d'ajouter une jointure gauche vers la table User selon le lien User.Id = Sale.IdClient.

```
1 SELECT
2     Sale.*,
3     User.LastName
4 FROM Sale
5 WHERE SaleDate = '2020-02-14'
```

### Indice :

La seule chose à rajouter dans cette requête est la jointure gauche vers la table User, à placer au bon endroit.

## VIII. Les alias de table

### Objectifs

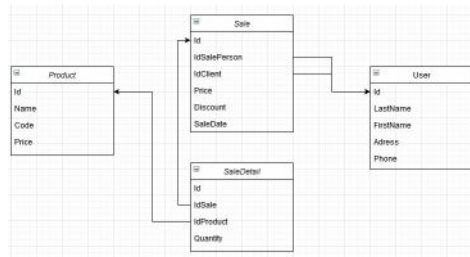
- Donner un alias à une table
- Distinguer les tables similaires entre elles dans une jointure

### Mise en situation

Comme nous l'avons vu, il est possible de faire des jointures sur plusieurs tables dans la même requête afin de récupérer toutes les informations dont on a besoin. Cependant, dans certains cas, il peut arriver que l'on ait besoin de joindre plusieurs fois sur la même table, ce qui peut poser des problèmes d'ambiguïté que l'on ne pourra pas régler en préfixant le champ par le nom de la table, puisque c'est la même table. Dans ce cas, nous allons devoir utiliser des **alias**.

## Un cas de jointures sur la même table

Grâce aux jointures, nous avons récupéré presque toutes les données de nos commandes en une seule requête : il ne nous reste plus qu'à rajouter les informations du vendeur et celles du client. D'après le schéma de base, les données du client et celles du vendeur sont présentes dans la table User :



Nous serions tentés de rajouter les deux jointures comme nous l'avons appris, en écrivant `JOIN User ON User.Id = Sale.IdSalePerson JOIN User ON User.Id = Sale.IdClient`, mais il y aurait une ambiguïté entre les deux `User.Id`.

Pour régler ce problème, il va falloir employer un alias, c'est-à-dire renommer temporairement notre table afin d'informer la base de données que la table User sera, une première fois, la table des vendeurs, et, une seconde fois, la table des clients.

### Syntaxe Définir un alias

Pour donner un alias à une table, il faut utiliser le mot-clé **AS** immédiatement après le nom de la table dans l'instruction de jointure.

```
1 SELECT
2     *
3 FROM Table1
4     INNER JOIN Table2 AS Alias
5         ON Alias.Id = Table1.reference2
```

Un alias peut être utilisé pour donner un nom alternatif à n'importe quelle table : il est requis pour lever des ambiguïtés, mais peut également être utilisé de manière facultative pour rendre une requête plus lisible.

### Exemple

Pour joindre les vendeurs et les clients à la table Sale, il suffit donc d'écrire :

```
1 SELECT
2     *
3 FROM Sale
4     INNER JOIN User AS Vendeur
5         ON Vendeur.Id = Sale.IdSalePerson
6     INNER JOIN User AS Client
7         ON Client.Id = Sale.IdClient
```

Une fois que l'on a donné un alias à nos tables, il faut utiliser leur alias dans la clause `ON`. Dans l'exemple ci-dessus, ce n'est pas la table User qui est mentionnée, mais son alias, Vendeur ou Client, selon le cas.

### Remarque

Cet exemple mélange volontairement des termes anglais issus de la base de données (Sale, SalePerson, User...) à des termes français (Vendeur, Client).

Ce n'est pas forcément une bonne pratique : il est préférable de rester homogène dans la rédaction d'une requête, mais permet ici de souligner l'utilisation des alias.

### Utiliser les alias dans un SELECT

Jusqu'à présent, nous avons utilisé la syntaxe `SELECT *` pour récupérer tous les champs, mais ce n'est pas une bonne pratique : il est préférable de lister uniquement les champs dont on a besoin. Maintenant que nos tables ont un alias qui leur est propre, il est possible de préciser quelle colonne retourner via la syntaxe `Table.Colonne`.

Il est même possible d'appliquer un alias pour renommer un nom de colonne, afin d'éviter que plusieurs colonnes aient le même nom.

#### Exemple

Si nous souhaitons récupérer la liste des noms des vendeurs et de ceux des clients associés, il faut écrire :

```
1 SELECT
2     Vendeur.Nom AS NomDuVendeur,
3     Client.Nom AS NomDuClient
4 FROM Sale
5     INNER JOIN User AS Vendeur
6         ON Vendeur.Id = Sale.IdSalePerson
7     INNER JOIN User AS Client
8         ON Client.Id = Sale.IdClient
```

Le résultat de cette requête aura deux colonnes : `NomDuVendeur` et `NomDuClient`.

#### Remarque

Le mot-clé `AS` est facultatif : si deux noms se succèdent entre deux mots-clés, le SGBD prendra le premier comme nom source et le second comme alias. `INNER JOIN User AS Vendeur` peut donc être remplacé par `INNER JOIN User Vendeur`.

La même remarque s'applique également pour les noms de colonnes : `Vendeur.Nom AS NomDuVendeur` peut être remplacé par `Vendeur.Nom NomDuVendeur`.

Toutefois, il est toujours bon de rester explicite dans l'écriture d'une requête, particulièrement lorsque celle-ci devient très longue.

#### Syntaxe À retenir

- Un alias est un second nom donné à une table ou une colonne dans une requête.
- Pour donner un alias, il est possible d'employer le mot-clé `AS`, ou simplement de faire suivre le nom de la table ou de la colonne par le nom de l'alias.

## IX. Exercice : Appliquez la notion

### Question

[solution n°6 p.24]

Une école de musique gère ses musiciens, leur groupe et leurs instruments grâce aux tables suivantes :

```
1 CREATE TABLE bands (
2     id INT PRIMARY KEY AUTO_INCREMENT,
3     name VARCHAR(100) NOT NULL,
4     genre VARCHAR(100) NOT NULL
5 );
6
7 CREATE TABLE instruments (
8     id INT PRIMARY KEY AUTO_INCREMENT,
9     name VARCHAR(100) NOT NULL
10 );
```

```

11
12 CREATE TABLE musicians (
13     id INT PRIMARY KEY AUTO_INCREMENT,
14     name VARCHAR(100) NOT NULL,
15     band_id INT NOT NULL,
16     instrument_id INT NOT NULL
17 );

```

Écrivez une requête permettant de récupérer le nom de tous les musiciens, celui de leurs instruments respectifs et celui de leur groupe de musique. Donnez un alias à ces trois champs afin de pouvoir les distinguer les uns des autres.

## X. Les clés de table

### Objectifs

- Pouvoir déterminer quels champs peuvent être utilisés comme clés primaires
- Distinguer les clés des index

### Mise en situation

Très souvent, une table comporte une **clé primaire**, c'est-à-dire une ou plusieurs colonnes dont la combinaison des valeurs fournit une référence unique pour chaque ligne de la table. Il est donc important de savoir quelles colonnes peuvent faire office de clés primaires, et de les définir en tant que telles.

#### Rappel Clé primaire

Une clé primaire est une valeur ou une combinaison de valeurs qui définit de façon unique chaque enregistrement d'une table. Les données de la table sont physiquement ordonnées selon la clé primaire de la table.

Dans MariaDB, une clé primaire est définie lors de la création de la table, soit par le mot-clé `PRIMARY KEY` à la suite du nom d'une colonne, soit en définissant la liste des colonnes grâce à l'instruction `PRIMARY KEY(colonne1, colonne2...)`.

```

1 -- Première méthode
2 CREATE TABLE Exemple1
3 (
4     Id int NOT NULL PRIMARY KEY,
5     Libelle varchar(50)
6 )
7 ;
8
9 -- Seconde méthode
10 CREATE TABLE Exemple2
11 (
12     Id1 int NOT NULL,
13     Id2 int NOT NULL,
14     Libele varchar(50),
15     PRIMARY KEY (Id1, Id2)
16 ;

```

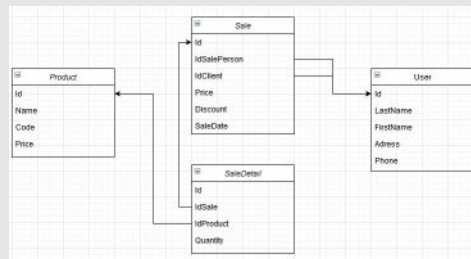


### Méthode

Les valeurs dans une table sont souvent divisées en deux catégories :

1. Les valeurs dites **métier**, c'est-à-dire les éléments qui vont être manipulés par les utilisateurs, comme un nom, un prix...
2. Les valeurs dites **techniques**, qui sont des valeurs utilisées uniquement par le programme, comme le compteur auto-incrémenté.

Si nous reprenons notre schéma de table, nous pouvons voir que la table des produits possède une valeur technique, l'identifiant, et trois valeurs métier.



Au moment de définir notre table, il a fallu déterminer quelle colonne utiliser en tant que clé primaire. Il est possible d'utiliser une valeur métier comme clé, on parle alors de clé naturelle. Par exemple, pour la table des produits, on aurait pu utiliser le code d'un produit plutôt qu'un compteur : c'est une valeur unique.

Le problème des valeurs métier est que l'on n'a jamais la certitude de leur unicité : elle dépend des utilisateurs finaux. A-t-on la garantie que le code produit ne sera jamais employé pour aucun autre produit, dans aucun magasin ? Que se passe-t-il si un mauvais code a été saisi et qu'il faut modifier tous les codes, et donc toutes les références à ce code ?

Une valeur technique est gérée directement par le SGBD, ce qui garantit leur unicité : bien qu'ils nécessitent de créer une colonne artificielle, il est préférable de les utiliser pour être une clé primaire.

### Exemple

Un piège récurrent de valeur métier utilisée comme clé primaire à mauvais escient est le numéro de sécurité sociale. On pourrait être tenté de penser qu'un tel numéro est unique, mais ce n'est pas vrai. Il existe plusieurs cas de figure où un numéro de sécurité sociale n'est pas une bonne clé primaire :

- Un mineur peut être rattaché au numéro de son parent : dans ce cas, deux personnes ont le même identifiant.
- Un individu peut recevoir un numéro temporaire : il aura alors plusieurs numéros de sécurité sociale dans sa vie. Dans ce cas, une même personne pourra avoir plusieurs identifiants.
- Il y a pu avoir une erreur de saisie : une personne s'est trompée et a donné un mauvais numéro.

En utilisant une valeur technique, toutes ces considérations n'ont plus lieu d'être.

Cependant, même si les valeurs métiers présumées uniques ne doivent pas être utilisées comme des clés primaires, il reste important de les identifier : elles peuvent être utilisées comme index.

### Rappel Les index

Lorsqu'un individu veut trouver une référence à un mot dans une encyclopédie, il n'a pas besoin de parcourir toutes les pages : il existe un index qui regroupe tous les termes importants présents dans le livre, avec les pages associées.

En base de données, il est possible d'utiliser le même principe et de définir une liste de valeurs issues d'une ou plusieurs colonnes de la table, qui serviront de critères de tri. Cela permettra aux requêtes filtrant ces colonnes d'être plus rapides.

Un index peut être unique, c'est-à-dire qu'il n'autorise qu'une seule ligne pour une combinaison de valeurs, ou non-unique. Une clé primaire, par définition, est un index unique.

Un index est créé par l'instruction `CREATE INDEX` suivie du nom de l'index, du nom de la table, et enfin de la liste des colonnes concernées.

#### Exemple

```
1 CREATE INDEX NomDeLIndex ON NomDeLaTable
2 (
3     Colonne1,
4     Colonne2,
5     Colonne3
6 );
```

### Lien entre clés, indexes et jointures

Bien que ça ne soit pas obligatoire, les jointures se font très fréquemment sur la clé primaire d'une autre table pour des raisons de performance.

En effet, lorsqu'une requête est envoyée à une base de données, celle-ci va tenter d'y répondre de la façon la plus efficace possible. Si elle doit renvoyer des valeurs selon certains critères, elle va devoir rechercher, parmi les différentes valeurs, lesquelles correspondent au critère souhaité. Si la base de données n'a pas d'autre solution, cela signifie qu'il faudra parcourir tout le contenu d'une table pour y trouver les valeurs correspondantes. Cela s'appelle un *table scan*.

Une jointure effectuée sur une clé primaire qui est un index va permettre d'éviter de parcourir tout le contenu : à l'image de la recherche d'un mot dans un dictionnaire, le SGBD pourra déterminer lorsqu'il n'a plus besoin de parcourir le reste de la table. Ainsi, si la jointure requiert de récupérer le produit n°5 dans une table de 1000 produits, mais que les identifiants sont triés par ordre croissant, une fois le produit n°6 trouvé, le SGBD sait qu'il peut arrêter sa recherche et retourner le résultat.

#### Syntaxe À retenir

- Une clé primaire est une colonne, ou un ensemble de colonnes, définissant chaque ligne de la table.
- Un index est une liste de valeurs issues d'une ou plusieurs colonnes permettant de trier les données.
- Les requêtes filtrant les valeurs d'une clé primaire ou d'un index sont bien plus performantes, car le moteur de la base de données n'a pas besoin de parcourir toute une table.

## XI. Exercice : Appliquez la notion

### Question

[solution n°7 p.24]

À partir de la table suivante, écrivez une instruction `CREATE INDEX` pour créer un index sur le couple (LastName, FirstName).

```
1 CREATE TABLE User
2 (
3     Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
4     LastName varchar(50) NOT NULL,
5     FirstName varchar(50) NOT NULL,
6     Address varchar(255) NOT NULL,
7     Phone varchar(20) NOT NULL
8 )
9 ;
```

**Indice :**

N'oubliez pas de donner un nom à l'index. Un index sur une table MaTable est souvent appelé `Idx_MaTable`, mais il peut prendre n'importe quel nom.

**XII. Essentiel**

Une jointure permet de relier deux tables entre elles dans le cadre d'une requête.

La syntaxe d'une jointure est :

```
1 SELECT
2   (Liste des colonnes)
3 FROM
4   Table1
5   (INNER / LEFT / RIGHT / OUTER) JOIN Table2
6   ON (liste des égalités)
```

Il est possible d'enchaîner autant de jointures que nécessaire.

Il est préférable que la condition des jointures porte sur une clé primaire ou sur un index de table, pour des raisons de performances.

**XIII. Auto-évaluation****A. Exercice final****Exercice 1**

[solution n°8 p.24]

Exercice

Le modèle relationnel permet...

- ☐ De conserver les données en vrac
- ☐ D'ordonner les données sur plusieurs tables
- ☐ D'éviter la redondance des données

Exercice

Une jointure permet de...

- ☐ Fusionner deux tables de façon à n'en avoir plus qu'une seule dans la base de données
- ☐ Créer une table virtuelle contenant les données de ces deux tables
- ☐ Créer une nouvelle table dans la base de données

Exercice

Une jointure gauche ramène toujours autant de lignes que contient la première table.

- ☐ Vrai
- ☐ Faux

Exercice

Peut-on joindre plus de deux tables ?

- ☐ Oui, au moyen d'une instruction `JOIN` par table
- ☐ Oui, au moyen d'une instruction `JOIN Table1 AND Table2 AND Table3...`
- ☐ Non, ce n'est pas possible

Exercice

La syntaxe recommandée pour une jointure est...

- ☐ `FROM Table1 JOIN Table2 ON Condition`
- ☐ `FROM Table1, Table2 WHERE Condition`

Exercice

Les jointures gauche et droite sont...

- ☐ Identiques, il possible de substituer `RIGHT` et `LEFT`
- ☐ Inversées, `A LEFT B` est égale à `B RIGHT A`

Exercice

Bien formater une requête un peu longue est...

- ☐ Indispensable, le SGBD utilise le formatage pour lire la requête
- ☐ Utile, afin de pouvoir lire plus facilement la requête par la suite
- ☐ Gênant, car les espaces supplémentaires augmentent la taille de la requête et diminuent les performances

Exercice

Peut-on joindre une même table à deux reprises ?

- ☐ Non, une table est unique
- ☐ Oui, il suffit de faire deux jointures
- ☐ Oui, à condition de donner un alias à au moins une des deux tables

Exercice

Peut-on sélectionner plusieurs fois la même colonne dans un `SELECT` ?

- ☐ Non
- ☐ Oui
- ☐ Oui, à condition de donner un alias à au moins une des deux colonnes

Exercice

Un index permet d'améliorer les performances...

- ☐ En multipliant les capacités du moteur
- ☐ En permettant au moteur d'effectuer moins d'opérations

## B. Exercice : Défi

Il est temps d'écrire la requête complète permettant au magasin d'obtenir le détail de ses ventes.

Voici la définition des tables composant la base de données :

```

1 CREATE TABLE Product
2 (
3     Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
4     Name varchar(50) NOT NULL,
5     Code varchar(10) NOT NULL,
6     Price decimal(10,2) NOT NULL
7 )
8 ;
9 CREATE TABLE User
10 (
11     Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
12     LastName varchar(50) NOT NULL,
13     FirstName varchar(50) NOT NULL,
14     Address varchar(255) NOT NULL,
15     Phone varchar(20) NOT NULL
16 )
17 ;
18 CREATE TABLE Sale
19 (
20     Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
21     IdSalePerson int NOT NULL,
22     IdClient int,
23     Price decimal(10,2) NOT NULL,
24     Discount decimal(10,2) NOT NULL,
25     SaleDate date
26 )
27 ;
28 CREATE TABLE SaleDetail
29 (
30     Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
31     IdSale int NOT NULL,
32     IdProduct int NOT NULL,
33     Quantity int
34 )
35 ;

```

### Question

[solution n°9 p.26]

Écrivez la requête permettant de ramener les colonnes Sale.Price, Sale.Discount, Sale.Date, Product.Name, Product.Price, SaleDetail.Quantity,

ainsi que le nom (user.LastName) à la fois du vendeur et du client, si celui-ci existe.

### Indice :

La table pivot est la table Sale, il faut commencer par elle.

### Indice :

Il est préférable de rajouter ensuite SaleDetail.

### Indice :

SaleDetail et Product vont ensemble.

### Indice :

User va devoir être jointe deux fois avec des alias.

**Indice :**

Les différentes jointures peuvent être de type `INNER` ou `LEFT` au choix, à une exception.

**Solutions des exercices**

#### p. 4 Solution n°1

En externalisant les données des participants, on se retrouve avec deux tables :

Réservations					Participants			
id	NomActivite	LieuActivite	Date	IdParticipant	id	NomParticipant	PrenomParticipant	TelephoneParticipant
1	Planche à voile	Plage	2030-07-06	1	1	Jean	Dupont	0123456789
2	Skateboard	Skatepark	2030-07-06	2	2	John	Doe	0147852369
3	Planche à voile	Plage	2030-07-07	3	3	Laure	Mondi	0213467958
4	VTT	Forêt	2030-07-08	1				
5	VTT	Forêt	2030-07-08	3				

#### p. 5 Solution n°2

Réservations				Participants			
id	Date	IdActivite	IdParticipant	id	NomParticipant	PrenomParticipant	TelephoneParticipant
1	2030-07-06	1	1	1	Jean	Dupont	0123456789
2	2030-07-06	2	2	2	John	Doe	0147852369
3	2030-07-07	1	3	3	Laure	Mondi	0213467958
4	2030-07-08	3	1				
5	2030-07-08	3	3				

Activites		
id	NomActivite	LieuActivite
1	Planche à voile	Plage
2	Skateboard	Skatepark
3	VTT	Forêt

On pourrait être tenté de créer une table des dates, étant donné que les dates sont dupliquées également, mais créer une table simplement pour centraliser une valeur n'est pas forcément utile.

#### p. 9 Solution n°3

Pour récupérer les informations des étudiants et des classes, il va falloir faire une jointure entre la table Etudiant et la table Classe. Or, dans la table Etudiant, il y a une colonne contenant l'ID de la classe de l'étudiant : `IdClasse`. Cette colonne correspond à la colonne `Id` de la table Classe. Nous allons donc pouvoir faire une jointure entre Etudiant et Classe sur les colonnes `IdClasse` et `Id`.

#### p. 9 Solution n°4

Il faut joindre les tables Classe (sur la colonne `Id`) et MaterielAlloue (sur la colonne `IdClasse`) pour avoir accès à la quantité. Il faut également joindre MaterielAlloue (colonne `IdMateriel`) et Materiel (colonne `Id`) pour avoir le nom de chaque matériel.

#### p. 13 Solution n°5

```

1 SELECT
2     Sale.*,
3     User.LastName
4 FROM Sale
5     LEFT JOIN User ON User.Id = Sale.IdClient
6 WHERE SaleDate = '2020-02-14'
```

p. 15 Solution n°6

```
1 SELECT
2     musicians.name as musicianName, bands.name AS band, instruments.name AS instrument
3 FROM
4     musicians
5 JOIN
6     bands ON musicians.band_id = bands.id
7 JOIN
8     instruments ON musicians.instrument_id = instruments .id
```

p. 18 Solution n°7

```
1 CREATE INDEX Idx_User ON User
2 (
3     LastName,
4     FirstName
5 );
```

Exercice p. 19 Solution n°8

**Exercice**

Le modèle relationnel permet...

- ☐ De conserver les données en vrac  
*En soi, ce n'est pas impossible, mais fortement déconseillé.*
- ☒ D'ordonner les données sur plusieurs tables  
*C'est le principe même de ce modèle.*
- ☐ D'éviter la redondance des données  
*Malheureusement, non, ce n'est pas toujours possible.*

**Exercice**

Une jointure permet de...


- ☐ Fusionner deux tables de façon à n'en avoir plus qu'une seule dans la base de données  
*Une jointure ne touche pas à la structure physique des données. Les tables sources restent présentes et ne sont pas altérées.*
- ☒ Créer une table virtuelle contenant les données de ces deux tables  
*Il ne faut pas oublier que cette table virtuelle n'existe que le temps de la requête.*
- ☐ Créer une nouvelle table dans la base de données  
*Aucune table n'est réellement créée lors d'une jointure.*

**Exercice**

Une jointure gauche ramène toujours autant de lignes que contient la première table.

- ☐ Vrai
- ☒ Faux



-  Si une même ligne de la première table possède plusieurs équivalents dans la seconde, elle sera multipliée d'autant.

### Exercice

---


Peut-on joindre plus de deux tables ?

- ☒ Oui, au moyen d'une instruction `JOIN` par table
- ☐ Oui, au moyen d'une instruction `JOIN Table1 AND Table2 AND Table3 ...`  
*Il ne faut pas confondre avec les listes de conditions. Il faut un `JOIN` par table.*
- ☐ Non, ce n'est pas possible  
*C'est possible, car le résultat d'une jointure est considéré comme une table et peut donc être joint à son tour.*

### Exercice

---


La syntaxe recommandée pour une jointure est...

- ☒ `FROM Table1 JOIN Table2 ON Condition`
- ☐ `FROM Table1, Table2 WHERE Condition`
-  La syntaxe `JOIN ON` est plus explicite, elle permet de distinguer la condition de jointure d'une condition de filtre.

### Exercice

---

Les jointures gauche et droite sont...

- ☐ Identiques, il possible de substituer `RIGHT` et `LEFT`
- ☒ Inversées, `A LEFT B` est égale à `B RIGHT A`
-  Il n'y aurait pas besoin de deux mots-clés s'ils étaient parfaitement identiques. De manière générale, c'est `LEFT` qui est le plus utilisé.

### Exercice

---

Bien formater une requête un peu longue est...

- ☐ Indispensable, le SGBD utilise le formatage pour lire la requête  
*Non. Le SGBD n'a besoin que des mots-clés, des virgules et des points-virgules.*
- ☒ Utile, afin de pouvoir lire plus facilement la requête par la suite  
*De manière générale en informatique, le code est lu bien plus souvent qu'il n'est écrit, c'est une bonne pratique.*
- ☐ Gênant, car les espaces supplémentaires augmentent la taille de la requête et diminuent les performances  
*Fort heureusement, non, le SGBD est parfaitement capable de lire une requête longue.*

### Exercice

---

Peut-on joindre une même table à deux reprises ?

- ☐ Non, une table est unique
- ☐ Oui, il suffit de faire deux jointures
- ☒ Oui, à condition de donner un alias à au moins une des deux tables

- Q C'est possible, à condition de renommer l'une des deux tables (ou les deux) avec des alias. Par contre, il ne peut y avoir deux fois le même alias.

### Exercice

Peut-on sélectionner plusieurs fois la même colonne dans un `SELECT` ?

- ☐ Non
- ☒ Oui
- ☐ Oui, à condition de donner un alias à au moins une des deux colonnes

- Q Oui et sans conditions. Ce sont les tables qui doivent recevoir un alias. Pour les colonnes, l'alias est une possibilité, mais pas une obligation.

### Exercice

Un index permet d'améliorer les performances...

- ☐ En multipliant les capacités du moteur
- ☒ En permettant au moteur d'effectuer moins d'opérations

- Q Lorsque le moteur parcourt un index, il peut se baser sur le tri de cet index pour éviter d'avoir à lire toutes les données. Autrement dit, il effectue moins d'opérations de lecture et trouve plus rapidement la réponse.

### p. 21 Solution n°9

```

1 SELECT
2     S.Price,
3     S.Discount,
4     S.SaleDate,
5     P.Name,
6     P.Price,
7     SD.Quantity,
8     Vendor.LastName AS Vendor_LastName,
9     Customer.LastName AS Customer_LastName
10 FROM Sale AS S
11     INNER JOIN SaleDetail AS SD
12         ON SD.IdSale = S.Id
13     INNER JOIN Product AS P
14         ON P.Id = SD.IdProduct
15     INNER JOIN User AS Vendor
16         ON Vendor.Id = S.IdSalePerson
17     INNER JOIN User AS Customer
18         ON Customer.Id = S.IdClient
19 ;

```