

## Tri de données avec le langage SQL

---

### Introduction :

Nous savons ce qu'est une base de données relationnelle, nous savons aussi la mettre en œuvre et l'utiliser grâce au SGBD et au langage SQL. Nous achevons ce chapitre sur les bases de données avec un cours destiné à compléter nos connaissances sur les requêtes SQL.

Il s'agira ici d'apprendre à formuler des recherches plus élaborées, faisant appel à des fonctions de tri ou de calcul. Nous verrons également comment articuler une requête SQL touchant plusieurs tables à la fois. Commençons donc par nous pencher sur les fonctions de calcul et transformation auxquelles il est possible d'avoir recours dans une commande SQL.





## 1 | Calculs et transformations dans une requête

À l'occasion d'une requête, il est possible de demander le résultat d'un calcul opéré sur la valeur d'une ou plusieurs colonnes pour chaque ligne concernée. On peut aussi demander le résultat d'un calcul qui porte sur l'ensemble des lignes d'une table. D'autres manipulations sont également possibles.

### a. Calcul portant sur la valeur d'une colonne

Le résultat d'un calcul effectué sur la valeur d'une colonne est stocké dans une colonne qui vient s'ajouter à la table résultat de la requête. Ainsi, on peut effectuer des calculs impliquant une ou plusieurs colonnes en ayant recours aux opérateurs arithmétiques suivants :



	soustraction
	multiplication
	division
	reste dans la division



## Exemple

Nous voulons la liste des recettes du club, avec leur nom et leur durée en seconde. Il est donc nécessaire d'effectuer un calcul avec la colonne *duree* qui est exprimée et stockée en minutes dans la base de données. Voici la commande SQL qui en résulte :

```
SELECT nomrecette, (duree * 60) AS dureeSecondes
FROM recette;
```



## À retenir

La clause **AS** permet de baptiser le nom de la nouvelle colonne destinée à recueillir le résultat du calcul.

Voici la table recette :

idRecette	nomRecette	niveau	duree	categorie	idAdhere
1	Mayonnaise	difficile	15	sauce	7
2	Omelette	facile	5	plat	6
3	Pâte brisée	moyen	20	autre	6
4	Quatre-quarts	facile	10	dessert	4

5	Purée de carottes	moyen	15	accompagnement	2
6	Poulet rôti	facile	15	plat	2
7	Sole meunière	difficile	20	plat	7

Le résultat obtenu est une table de deux colonnes :

nomRecette	dureeSecondes
Mayonnaise	900
Omelette	300
Pâte brisée	1200
Quatre-quarts	600
Purée de carottes	900
Poulet rôti	900
Sole meunière	1200

### b. Calcul portant sur l'ensemble des lignes d'une table

Entre autres calculs et transformations, penchons-nous sur les fonctions statistiques simples que l'on peut appeler avec SQL, recensées dans le tableau qui suit :

COUNT(*)	Nombre de lignes dans la table
----------	--------------------------------

MAX(colonnex)	valeur maximum dans la colonne
MIN(colonnex)	valeur minimum dans la colonne
AVG(colonnex)	valeur moyenne de la colonne
SUM(colonnex)	somme des valeurs de la colonne



## Exemple

Si l'on veut connaître la durée moyenne des recettes du club de cuisine, en minutes, on passera la commande SQL suivante au SGBD :

```
SELECT AVG(duree) AS dureeMoyenne
FROM recette;
```

Le résultat alors retourné est une table d'une colonne et d'une ligne :

dureeMoyenne
14.2857

Et si l'on souhaite connaître le nombre d'adhérent·e·s du club de cuisine, il suffira d'entrer la commande suivante :

```
SELECT COUNT(*) AS nbAdherents
FROM adherent;
```

Voici la table *adherent* :

idAdherent	nomAdherent	prenom	mail	nu
1	DURAND	Jacques	j.durand@orange.fr	16

2	DUVAL	Paul	paul.d@sfr.fr	135
3	DELORS	Sophie	sodelors@outlook.com	22
4	VERGER	Jehan	jverger@aol.com	4t
5	MONVILLE	Fleur	fleur1212@free.fr	9
6	BRIARD	Paul	paul.briard27@hotmail.fr	19
7	LÉGER	Marguerite	mag0506@perso.com	17
8	FLAMAND	Lise	flise@gmail.com	32

Et voici le résultat retourné, qui est une table d'une colonne et d'une ligne :

nbAdherents
8



On remarquera au passage qu'on ne spécifie pas une colonne en particulier dans la fonction `COUNT` puisqu'il s'agit de compter le nombre de toutes les lignes dans la table, sans s'intéresser à une colonne en particulier.

## c. Autres manipulations sur les données

D'autres transformations sur les colonnes sont possibles lors d'une requête d'interrogation, comme par exemple le passage en majuscule d'une colonne de type chaîne de caractères avec la fonction `UPPER`. On peut également demander l'extraction du mois d'une date grâce à la fonction

`MONTH`.



### Exemple

Voici la requête permettant d'obtenir la liste des numéros, noms et prénoms des adhérent·e·s affichés en majuscules de notre club de cuisine :

```
SELECT idAdherent, UPPER(nomAdherent) AS nomMaj, UPPER(prenom) AS  
prenomMaj  
FROM adherent;
```

Le résultat est une table de trois colonnes :

idAdherent	nomMaj	prenomMaj
1	DURAND	JACQUES
2	DUVAL	PAUL
3	DELORS	SOPHIE
4	VERGER	JEHAN
5	MONVILLE	FLEUR
6	BRIARD	PAUL
7	LEGER	MARGUERITE
8	FLAMAND	LISE

Il peut être parfois intéressant d'effectuer des calculs statistiques portant sur des groupes de lignes et toutes les lignes d'une table. On aura alors recours à des opérations d'agrégation. Nous allons voir également que l'on peut exiger d'une requête qu'elle nous restitue sa table résultat de manière triée.

## 2 | Agrégats et tri

### a. Calcul sur agrégats

On effectue une opération dite d'**agrégation** pour regrouper les lignes d'une table par la même valeur qu'elles contiennent dans une colonne. L'objectif d'effectuer une agrégation est, en général, de pratiquer des calculs statistiques sur les groupes ainsi constitués.



La clause employée pour pratiquer une agrégation est **GROUP BY** :

```
SELECT nomColonne1, fonction_statistique
FROM nomTable
GROUP BY nomColonne1;
```



Notre club de cuisine souhaite connaître le nombre de recettes dans sa base pour chaque catégorie.

La table des recettes est la suivante :

idRecette	nomRecette	niveau	duree	categorie	idAdh
1	Mayonnaise	difficile	15	sauce	7
2	Omelette	facile	5	plat	6
3	Pâte brisée	moyen	20	autre	6

4	Quatre-quarts	facile	10	dessert	4
5	Purée de carottes	moyen	15	accompagnement	2
6	Poulet rôti	facile	15	plat	2
7	Sole meunière	difficile	20	plat	7

Il suffira donc de transmettre la requête SQL suivante au SGBD :

```
SELECT categorie, COUNT(*) AS nombre
```

```
FROM recette
```

```
GROUP BY categorie;
```

La table résultat qui nous sera alors retournée sera par conséquent :

categorie	nombre
sauce	1
plat	3
autre	1
dessert	1
accompagnement	1

## b. Restriction sur le résultat d'un calcul sur agrégat



On peut ajouter une opération de filtrage à la table résultant d'une opération sur agrégat. La clause `HAVING` permet d'exprimer le critère de filtrage sur cette table résultat.



Le club décide de recueillir les seules catégories de recette dont la durée moyenne est au moins de 14 minutes.

La commande SQL suivante nous permet de calculer la durée moyenne des recettes par catégorie :

```
SELECT categorie, AVG(duree) AS dureeMoyenne
FROM recette
GROUP BY categorie;
```

La table résultat qui est restituée est ainsi :

<i>categorie</i>	<i>dureeMoyenne</i>
sauce	15.0000
plat	13.3333
autre	20.0000
dessert	10.0000
accompagnement	15.0000

En complétant la commande par la clause `HAVING dureeMoyenne >= 14`, on ne retient que les catégories figurant dans la table résultat dont la colonne *dureeMoyenne* répond au critère du filtre :

```
SELECT categorie, AVG(duree) AS dureeMoyenne
FROM recette
HAVING dureeMoyenne >= 14;
```

GROUP BY *categorie*

HAVING *dureeMoyenne* >= 14;

Le résultat qui est restitué ne nous étonne pas :

categorie	dureeMoyenne
sauce	15.0000
autre	20.0000
accompagnement	15.0000

### c. Tris

Lorsque l'on effectue une requête, par défaut, la table résultat qui est restituée n'est pas triée.

 À retenir

Pour que ce soit le cas, on emploie le mot-clé **ORDER BY** suivi de la ou les colonnes sur lesquelles doit porter le tri.

 Astuce

Si plusieurs colonnes sont spécifiées dans la consigne de tri, alors le tri s'effectuera d'abord sur les valeurs de la colonne la plus à gauche, puis selon les valeurs des colonnes données pour le tri de gauche à droite.

Notons par ailleurs que le tri s'effectue par défaut de manière ascendante. L'option **DESC** inversera le tri pour le critère concerné.

 Exemple

Notre club de cuisine souhaite avoir la liste de ses adhérent·e·s constituée de leurs numéros, noms et prénoms mis en majuscules, triée par nom, puis par prénom.

Reprenons la table des adhérent·e·s :

idAdherent	nomAdherent	prenom	mail
1	DURAND	Jacques	j.durand@orange.fr
2	DUVAL	Paul	paul.d@sfr.fr
3	DELORS	Sophie	sodelors@outlook.com
4	VERGER	Jehan	jverger@aol.com
5	MONVILLE	Fleur	fleur1212@free.fr
6	BRIARD	Paul	paul.briard27@hotmail.fr
7	LÉGER	Marguerite	mag0506@perso.com
8	FLAMAND	Lise	flise@gmail.com

Voici la requête qui répond à l'objectif visé :

```
SELECT idAdherent, UPPER(nomAdherent), UPPER(prenom)
FROM adherent
```

ORDER BY *nomAdherent, prenom;*

Le résultat suivant nous est restitué :

idAdherent	nomMaj	prenomMaj
6	BRIARD	PAUL
3	DELORS	SOPHIE
1	DURAND	JACQUES
2	DUVAL	PAUL
8	FLAMAND	LISE
7	LEGER	MARGUERITE
5	MONVILLE	FLEUR
4	VERGER	JEHAN

Nous savons maintenant effectuer, dans une requête, des calculs sur les colonnes et sur les lignes d'une table ou transformer une colonne, ainsi que trier le résultat d'une requête. Voyons maintenant quand et comment effectuer des requêtes portant sur plusieurs tables.

### 3 | Requetes portant sur plusieurs tables

#### a. Contexte d'une requête portant sur plusieurs tables

Dans une base de données existent quasi systématiquement des **associations**. Celles-ci établissent un lien entre des entités. Ainsi, pour retrouver des informations relatives à une même entité, on peut être amené à récupérer celles-ci dans plusieurs tables par le biais de ces associations.

Pour rappel, les tables *adherent* et *ville* sont constituées ainsi :

idAdherent	nomAdherent	prenom	mail
1	DURAND	Jacques	j.durand@orange.fr
2	DUVAL	Paul	paul.d@sfr.fr
3	DELORS	Sophie	sodelors@outlook.com
4	VERGER	Jehan	jverger@aol.com
5	MONVILLE	Fleur	fleur1212@free.fr
6	BRIARD	Paul	paul.briard27@hotmail.fr
7	LÉGER	Marguerite	mag0506@perso.com
8	FLAMAND	Lise	flise@gmail.com

idVille	nomVille	codePostal
1	LE HAVRE	76600
2	CAEN	14000

3	NANTES	44000
4	RENNES	35000
5	QUIMPER	29000
6	VANNES	56000
7	SAINT-BRIEUC	22000
8	PONTIVY	56300
9	LOUDEAC	22600
10	QUIMPERLE	29300

Nous ne sommes pas en mesure de connaître, dans notre table *adherent*, le nom de la ville où chaque adhérent·e habite. Nous connaissons néanmoins le numéro d'identification, soit *idVille*, de la ville où chacun habite. Grâce à ce numéro, nous pouvons retrouver le nom de celle-ci dans une autre table : la table *ville*.



Exemple

Ainsi, si nous souhaitons recueillir la liste des adhérent·e·s de notre club de cuisine, constituée de leur numéro d'identification, nom, prénom et ville où ils résident, nous sommes appelés à effectuer des recherches dans deux tables : *adherent* et *ville*.

Dans ce contexte où nous effectuons une requête portant sur plusieurs tables, une ambiguïté concernant la désignation de leurs colonnes respectives peut se présenter.

→ En effet, rien n'interdit de construire une base de données où **un même nom de colonne peut figurer dans deux tables différentes.**



Exemple

Dans notre base de données, la table *recette* contient une colonne *idRecette*, tout comme la table *etape*.



Aussi, pour lever toute ambiguïté dans une requête qui porterait sur des tables ayant des colonnes nommées de la même manière, il est alors nécessaire de préfixer le nom de la colonne par celui de la table concernée.

→ Finalement, on préférera, qu'il y ait ambiguïté ou non, préfixer tous les noms de colonnes du nom de la table à laquelle celles-ci appartiennent.



Dans notre exemple, on désignera nos champs *idRecette* ainsi :

- *recette.idRecette* ;
- *etape.idRecette* ;

Et voici une requête utilisant un préfixe de nom de table :

```
SELECT recette.nomRecette, recette.categorie, recette.duree  
FROM recette;
```

À l'usage, on réalise que ce préfixe peut alourdir considérablement la rédaction de certaines requêtes, déjà complexes par ailleurs.

→ On peut alors faire le choix de substituer le préfixe du nom de la table par un alias, plus court, qui sera précisé dans la commande SQL avec le mot-clé **AS**.

Notons que le recours à l'alias n'est pas obligatoire. .



En affectant l'alias « re » pour notre table *recette*, la commande précédente est ainsi reformulée de façon plus compacte :

```
SELECT re.nomRecette, re.categorie, re.duree  
FROM recette AS re;
```

Apprenons maintenant à formuler des requêtes portant sur plusieurs tables. Il s'agit de l'opération de **jointure** que nous allons dès à présent détailler.

### b. La jointure interne

Dans le cas de la base de données de notre club de cuisine, pour obtenir la liste des adhérent·e·s constituée de leur numéro d'adhérent, leur nom, leur prénom et de la ville où ils résident, nous pourrions formaliser la requête suivante :

```
SELECT ad.idAdherent, ad.nomAdherent, ad.prenom, vi.nomVille  
FROM adherent AS ad, ville AS vi  
WHERE ad.idVille = vi.idVille
```

La logique de cette requête est tout à fait correcte. Le SGBD l'exécutera et donnera le résultat suivant :

idAdherent	nomAdherent	prenom	nomVille
1	Durand	Jacques	LE HAVRE
2	Duval	Paul	RENNES
3	Delors	Sophie	NANTES
4	Verger	Jehan	QUIMPER
5	Monville	Fleur	CAEN
6	Briard	Paul	RENNES



7	Leger	Marguerite	NANTES
8	Flamand	Lise	CAEN



Attention

Cependant, il est nettement préférable d'avoir recours à un opérateur de jointure, nommé **JOIN**, spécifique aux requêtes portant sur plusieurs tables.

→ Le SGBD traitera alors la requête de manière optimisée.



Définition

## Jointure :

Une jointure consiste à combiner des données issues de deux ou plusieurs tables, sur la base d'un point commun.

La syntaxe requise pour effectuer une jointure est la suivante, dans le cadre de notre exemple :

```
SELECT ad.idAdherent, ad.nomAdherent, ad.prenom, vi.nomVille
FROM adherent AS ad JOIN ville AS vi ON ad.idVille = vi.idVille;
```

Cette requête restituera le même résultat que précédemment, mais le SGBD aura optimisé les moyens de fournir ce résultat.

Notons par ailleurs qu'il est tout à fait possible de formaliser une requête ;SQL portant sur plus de deux tables.



Exemple

Nous souhaitons avoir la liste des ingrédients de toutes les recettes, constituée de l'identifiant de la recette, du nom de la recette et du nom de l'ingrédient. La table *utilise* nous permet de savoir quelle recette utilise quel ingrédient. En revanche, dans cette table, les recettes sont identifiées par la clé *idRecette*, ce qui ne nous donne pas leur nom. Idem

pour la clé *idIngredient* qui ne nous permet pas de connaître le nom des ingrédients.

Mais on peut retrouver le nom chaque recette dans la table *recette* avec la clé *idRecette*. De même, on retrouve le nom de chaque ingrédient dans la table *ingredient* avec la clé *idIngredient*.

Voici nos tables :

idIngredient	idRecette	quantite	unite
1	1	1	jaune d'oeuf
1	2	3	oeufs entiers
1	4	3	oeufs entiers
2	1	1	cuillère
3	1	1	filet
4	1	10	cL
5	2	1	pincée
5	3	1	cuillère à café
5	4	1	pincee
5	7	1	pincée
6	2	2	tour de moulin
6	7	2	tours de moulin
7	2	30	g
8	2	2	cuillères à soupe
8	6	1	cuillère à soupe

8	7	2	cuillères à soupe
9	3	150	g
9	4	180	g
9	7	100	g
10	3	75	g
10	4	180	g
10	7	20	g
11	4	180	g
12	3	20	cL
13	5	500	g
14	5	150	g
15	5	2	pincées
16	5	1	pincée
17	6	1	poulet entier
18	7	2	branches
19	7	1	fruit entier
20	7	4	filets

idRecette	nomRecette	niveau	duree	categorie	idAdh
1	Mayonnaise	difficile	15	sauce	7
2	Omelette	facile	5	plat	6

3	Pâte Brisée	moyen	20	autre	6
4	Quatre-quarts	facile	10	dessert	4
5	Purée de carottes	moyen	15	accompagnement	2
6	Poulet rôti	facile	15	plat	2
7	Sole meunière	difficile	20	plat	7

idIngredient	nomIngredient	type
1	oeuf	viande
2	moutarde	condiment
3	vinaigre	condiment
4	huile	condiment
5	sel	condiment
6	poivre	condiment
7	gruyère râpé	crèmerie
8	huile d'olive	condiment
9	farine	épicerie
10	beurre	crèmerie
11	sucré	épicerie
12	lait	crèmerie

13	carotte	légume
14	crème fraîche	crèmerie
15	cumin	condiment
16	muscade	condiment
17	poulet	viande
18	persil	condiment
19	citron	fruit
20	sole	viande

Nous pourrions rédiger la commande SQL suivante :

```
SELECT r.idRecette, r.nomRecette, i.nomIngredient
FROM utilise AS u, recette AS r, ingredient AS i
WHERE u.idRecette = r.idRecette AND u.idIngredient = i.idIngredient
ORDER BY r.idRecette, i.nomIngredient;
```

Mais le SGBD n'optimisera pas ses traitements pour restituer le résultat. Utilisons à nouveau une jointure avec le mot-clé **JOIN** :

```
SELECT r.idRecette, r.nomRecette, i.nomIngredient
FROM utilise AS u JOIN recette AS r JOIN ingredient AS i ON (u.idRecette = r.idRecette)
AND (u.idIngredient = i.idIngredient)
ORDER BY r.idRecette, i.nomIngredient;
```

Le résultat de ces deux commandes sera identique :

idRecette	nomRecette	nomIngredient
1	Mayonnaise	huile

1	Mayonnaise	moutarde
1	Mayonnaise	oeuf
1	Mayonnaise	vinaigre
2	Omelette	gruyère râpé
2	Omelette	huile d'olive
2	Omelette	oeuf
2	Omelette	poivre
2	Omelette	sel
3	Pâte brisée	beurre
3	Pâte brisée	farine
3	Pâte brisée	lait
3	Pâte brisée	sel
4	Quatre-quarts	beurre
4	Quatre-quarts	farine
4	Quatre-quarts	oeuf
4	Quatre-quarts	sel
4	Quatre-quarts	sucré
5	Purée de carottes	carotte
5	Purée de carottes	crème fraîche
5	Purée de carottes	cumin
5	Purée de carottes	muscade

6	Poulet rôti	huile d'olive
6	Poulet rôti	poulet
7	Sole meunière	beurre
7	Sole meunière	citron
7	Sole meunière	farine
7	Sole meunière	huile d'olive
7	Sole meunière	persil
7	Sole meunière	poivre
7	Sole meunière	sel
7	Sole meunière	sole

### c. La jointure externe

La jointure externe permet d'effectuer une jointure en incluant les lignes dont la clé étrangère qui fait le lien entre deux tables n'est pas renseignée. Sachant que deux tables sont considérées dans cette jointure, il faut préciser laquelle des deux servira de **référence**.

Ainsi, toutes les lignes de la table qui servira de référence seront représentées dans la table résultat. L'attribut ou les attributs récupérés via la clé étrangère seront renseignés avec la valeur trouvée dans la seconde table, ou renseignée à « NULL » lorsque la clé étrangère n'est pas renseignée dans la première table.



À retenir

Il faudra employer pour cela la clause `OUTER JOIN`.



Exemple

On souhaite avoir la liste des adhérent·e·s constituée de leur numéro d'identification, leur nom, leur prénom, l'identifiant et le nom de chaque recette rédigée par un adhérent donné.

On passera la commande suivante :

```
SELECT a.idAdherent, a.nomAdherent, a.prenom, r.idRecette, r.nomRecette
FROM adherent AS a JOIN recette AS r ON a.idAdherent = r.idAdherent
ORDER BY a.idAdherent, r.nomRecette;
```



Pour rappel, les tables concernées sont constituées comme suit :

idRecette	nomRecette	niveau	duree	categorie	idAdh
1	Mayonnaise	difficile	15	sauce	7
2	Omelette	facile	5	plat	6
3	Pâte brisée	moyen	20	autre	6
4	Quatre-quarts	facile	10	dessert	4
5	Purée de carottes	moyen	15	accompagnement	2
6	Poulet rôti	facile	15	plat	2
7	Sole meunière	difficile	20	plat	7

idAdherent	nomAdherent	prenom	mail
------------	-------------	--------	------



1	DURAND	Jacques	j.durand@orange.fr
2	DUVAL	Paul	paul.d@sfr.fr
3	DELORS	Sophie	sodelors@outlook.com
4	VERGER	Jehan	jverger@aol.com
5	MONVILLE	Fleur	fleur1212@free.fr
6	BRIARD	Paul	paul.briard27@hotmail.fr
7	LÉGER	Marguerite	mag0506@perso.com
8	FLAMAND	Lise	flise@gmail.com

On obtient :

idAdherent	nomAdherent	prenom	idRecette	nomRecette
2	Duval	Paul	6	Poulet rôti
2	Duval	Paul	5	Purée de carottes
4	Verger	Jehan	4	Quatre-quarts

6	Briard	Paul	2	Omelette
6	Briard	Paul	3	Pâte brisée
7	Leger	Marguerite	1	Mayonnaise
7	Leger	Marguerite	7	Sole meunière

Mais nous souhaiterions que cette liste d'adhérent·e·s n'exclue pas ceux qui n'ont rédigé aucune recette. Pour cela, il faut faire appel à une **jointure externe**, en prenant *adherent* comme table de référence.

La commande SQL qui correspond se formalise ainsi :

```
SELECT a.idAdherent, a.nomAdherent, a.prenom, r.idRecette, r.nomRecette
FROM adherent AS a LEFT OUTER JOIN recette AS r ON a.idAdherent = r.idAdherent
ORDER BY a.idAdherent, r.nomRecette;
```

**LEFT** indique que c'est la table qui apparaît à gauche dans la clause **FROM** qui sert de référence.

Ce qui veut dire que la commande qui suit est identique à la précédente :

```
SELECT a.idAdherent, a.nomAdherent, a.prenom, r.idRecette, r.nomRecette
FROM adherent AS a RIGHT OUTER JOIN recette AS r ON a.idAdherent = r.idAdherent
ORDER BY a.idAdherent, r.nomRecette;
```

Et le résultat obtenu est le suivant :

idAdherent	nomAdherent	prenom	idRecette	nomRecette
1	Durand	Jacques	NULL	NULL
2	Duval	Paul	6	Poulet rôti
2	Duval	Paul	5	Purée de carottes

3	Delors	Sophie	NULL	NULL
4	Verger	Jehan	4	Quatre-quarts
5	Monville	Fleur	NULL	NULL
6	Briard	Paul	2	Omelette
6	Briard	Paul	3	Pâte brisée
7	Leger	Marguerite	1	Mayonnaise
7	Leger	Marguerite	7	Sole meunière
8	Flamand	Lise	NULL	NULL

Prenons maintenant la table recette en référence en inscrivant la commande :

```
SELECT a.idAdherent, a.nomAdherent, a.prenom, r.idRecette, r.nomRecette
FROM adherent AS a RIGHT OUTER JOIN recette AS r ON a.idAdherent = r.idAdherent
ORDER BY a.idAdherent, r.nomRecette;
```

On obtient alors :

idAdherent	nomAdherent	prenom	idRecette	nomRecette
2	Duval	Paul	6	Poulet rôti
2	Duval	Paul	5	Purée de carottes
4	Verger	Jehan	4	Quatre-quarts
6	Briard	Paul	2	Omelette
6	Briard	Paul	3	Pâte brisée

7	Leger	Marguerite	1	Mayonnaise
7	Leger	Marguerite	7	Sole meunière

Comme voulu, toutes les lignes de la table recette sont représentées dans la table résultat. Notons que l'on n'observe aucune colonne nulle (« NULL ») parmi les lignes de la table *résultat*. Cette particularité est due au fait que toute recette est forcément rédigée par un·e adhérent·e. L'inverse n'étant pas vrai, on l'a vu précédemment : tout adhérent n'a pas forcément rédigé une recette.

Pour terminer, on peut tenter de combiner différentes clauses. Si nous souhaitons par exemple connaître tous les adhérent·e·s qui n'ont pas proposé de recette, il suffira de passer la commande SQL suivante :

```
SELECT a.idAdherent, a.nomAdherent, a.prenom, r.idRecette, r.nomRecette
FROM recette AS r RIGHT OUTER JOIN adherent AS a ON a.idAdherent = r.idAdherent
WHERE r.idRecette IS NULL;
```

Ce qui donnera le résultat escompté, à savoir :

idAdherent	nomAdherent	prenom	idRecette	nomRecette
1	Durand	Jacques	NULL	NULL
3	Delors	Sophie	NULL	NULL
5	Monville	Fleur	NULL	NULL
8	Flamand	Lise	NULL	NULL

### Conclusion :

Nous achevons ainsi ce cours donnant des notions complémentaires sur le langage SQL. Vous savez dorénavant rédiger des requêtes élaborées, faisant appel à des mécanismes de calcul, de regroupement, de transformation, de tri ou encore de jointure.

