****

**Jointures en SQL**

Table des matières

[**Cours**](#_ylaz12tvtk5y) **3**

[Le besoin de jointures](#_kiddla7gzxyy) 3

[Alias pour les noms de tables](#_e0060aawgg1m) 4

[Les clefs primaires et étrangères](#_5j18lvh543s) 5

[Les différents types de jointures](#_c1yz7ozduhb1) 6

[INNER JOIN](#_gztgzzyz1rq0) 6

[OUTER JOIN](#_39i703m1vn8x) 7

[LEFT JOIN](#_zbztw5c2kqsk) 8

[Jointures et opérations](#_t3l1kbakv7vk) 9

[Multiples jointures](#_ueu2r7xtw151) 10

[**Annexe**](#_j858o8qfvbau) **11**

[Représentation ensembliste des jointures](#_3g2n2u4xjo99) 11

[**Résumé**](#_1hxc9p19k10t) **12**

[**Pour aller plus loin**](#_f07akaisw0cc) **13**

# Cours

## Le besoin de jointures

Nous avions introduit le modèle relationnel en début de cours en citant ses deux fondements: les tables et les relations. Nous avons étudié ensemble les tables et comment les interroger, les filtrer, les trier ou encore grouper leurs informations. Néanmoins si on respecte le principe de non redondance d’information, il est impossible de regrouper toutes les informations dans une seule et même table. On préfère au contraire créer plusieurs tables homogènes, c’est-à-dire contenant des données qui leur sont spécifiques. Par exemple, une table d’acheteurs, une table de produits et une table d’achats effectués. C’est pourquoi, il est indispensable de savoir relier les tables entre elles pour combiner leurs informations. L’action de relier des tables entre elles en SQL est appelée opération de **jointure**. Ils existent différents types de jointures mais elles reposent toutes sur la présence de clefs primaires et étrangères que nous définirons au chapitre suivant.

Pour le cours d'aujourd'hui, nous reprenons notre boutique de primeur sauf que cette fois-ci, nous avons 3 tables:

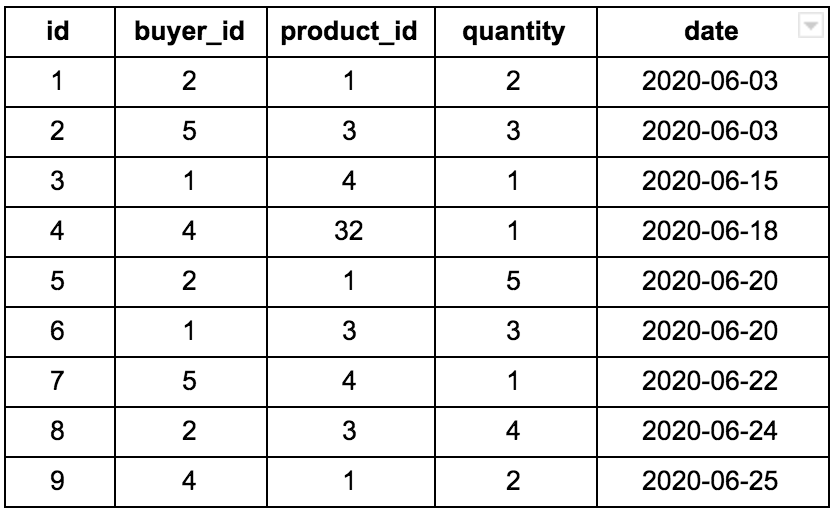
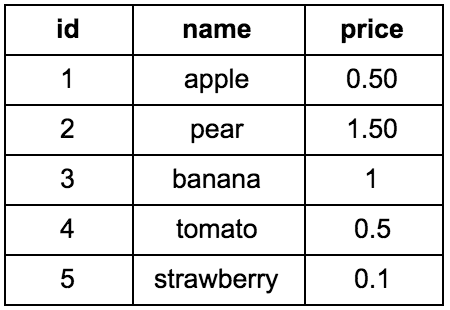


Table *purchases*

  
Table *products*

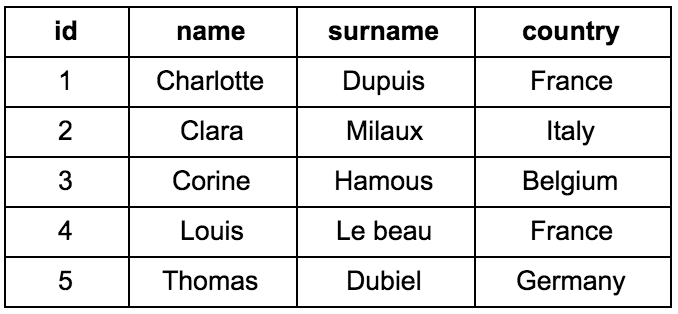


Table *buyers*

La table *purchases* regroupe tous les achats de produits effectués par les acheteurs avec la quantité achetée et la date d’achat.

La table *products* regroupe l’ensemble des produits qui sont vendus dans la boutique.

La table *buyers* regroupe l’ensemble des acheteurs enregistrés de la boutique.

Vous voyez ici que chaque table regroupe des informations bien distinctes entre elles, elles traitent chacune une catégorie de l’information, c’est le critère d’homogénéité.

## Alias pour les noms de tables

Nous avons vu l’utilisation d’alias dans la clause SELECT pour renommer une colonne des résultats en sortie et que cela n’affectait en rien le nom de cette colonne en base de données. Il est également possible d’utiliser des alias pour des noms des tables. Cela permet de faire référence à une table plus facilement quand son nom est long ou pour éviter toute ambiguïté quand nous ferons des jointures entre deux tables qui utiliseront les mêmes noms de colonnes. Voyons un exemple simple où nous souhaitons afficher seulement les prénoms des acheteurs:

| SELECT name  FROM buyers |
| --- |

Nous pouvons écrire cette même requête mais cette fois-ci en utilisant un alias pour le nom de la table buyers:

| SELECT b.name FROM buyers AS b |
| --- |

Comme vous pouvez le voir, après la clause FROM nous avons ajouté la clause AS pour indiquer que nous voulons faire référence à la table *buyers* non pas par son nom mais par son alias « b ». Ainsi dans la clause SELECT, pour ne renvoyer que la colonne name, nous devons écrire NOM\_DE\_L\_ALIAS.NOM\_COLONNE donc ici b.name pour y faire référence.

Les alias pour les tables sont très souvent utilisés et donc important à comprendre. Dans la suite de ce cours, nous utiliserons des alias dans les jointures.

## Les clefs primaires et étrangères

Mais avant de rentrer dans le cœur du sujet, il est très important aussi de comprendre la différence entre clefs primaires et clefs étrangères:

* Clef primaire: est la colonne de la table qui permet d’identifier de façon unique chacune des lignes. Il n’y a qu’une seule clef primaire par table. Dans de rares occasions, la clef primaire est l’association de plusieurs colonnes de la table.
* Clef étrangère: est la colonne de la table qui fait référence à la clé primaire d’une autre table. Une table peut contenir plusieurs clefs étrangères faisant référence à plusieurs clefs primaires de plusieurs autres tables.

Prenons l’exemple de la table *Purchases*. La clef primaire est la colonne **id**, elle permet d’identifier de façon unique chacun des achats réalisés. A chaque nouvel achat, une nouvelle ligne va être créée dans la table et le SGBD va automatiquement calculer l’id à insérer en regardant le dernier id de la table et en lui ajoutant +1.

Il y a également dans cette table deux colonnes qui sont deux clefs étrangères: **product\_id** et **buyer\_id**. En effet, elles font référence respectivement à la clef primaire **id** de la table *products* et à la clef primaire **id** de la table *buyers.*

Ainsi pour une ligne de la table *purchases*, via les clefs secondaires, vous avez aussi bien accès aux informations des acheteurs comme leurs noms et prénoms, mais aussi aux informations des produits comme leurs noms et leurs prix sans avoir besoin de rappeler toutes ces informations dans la table de départ. Pour cela, nous devons voir comment écrire en SQL les opérations de jointures.

## Les différents types de jointures

Il existe plusieurs façons de joindre deux tables entre elles. En effet, souhaitez-vous conserver toutes les informations de la première et ajouter les informations de la deuxième quand c’est possible ? Préférez vous ne garder que les lignes où il y a bien une correspondance entre les deux tables ? etc…   
Ci-dessous, nous allons détailler le fonctionnement de chacune d’entre elles.

Chaque opération de jointure cherche des correspondances entre une table gauche A et une table droite B. On indiquera toujours en SQL sur quelle(s) colonne(s) respectives il faut s’appuyer pour la correspondance. Remarque, elles n’ont pas l’obligation d’avoir le même nom dans les deux tables. Par ailleurs, l’ordre est très important ! En effet joindre B à A où joindre A à B n’est pas toujours équivalent ! Commençons par présenter la jointure la plus classique.

### INNER JOIN

L’INNER JOIN est le type de jointure par défaut dans la plupart des SGBD. Une jointure de type INNER JOIN ne va garder que les lignes de la table A qui ont trouvé une correspondance avec la table B sur la ou les colonnes indiquées. Les autres lignes seront simplement ignorées du résultat. Observons un exemple, nous cherchons à avoir les informations des produits achetés dans la table *purchases* et pas simplement leurs ids, pour cela nous allons procéder à une jointure de la table *purchases* à la table *products*:

| SELECT \* FROM purchases AS pu INNER JOIN products AS pr ON pu.product\_id = pr.id |
| --- |

Vous connaissez toutes les clauses jusqu’au FROM. A partir de là vous précisez juste après cette dernière le nom de la table avec laquelle vous souhaitez faire une correspondance, ici *products*. Puis avec la clause ON vous précisez sur quelles colonnes faire cette correspondance en nommant celle de la table *purchases* **product\_id** (clef secondaire) qui doit être égale à celle de la table *products* **id** (clef primaire). En procédant ainsi, vous obtiendrez les résultats suivants:

| **pu.id** | **pu.buyer\_id** | **pu.product\_id** | **pu.quantity** | **pu.date** | **pr.id** | **pr.name** | **pr.price** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 1 | 2 | 2020-06-03 | 1 | apple | 0.50 |
| 2 | 5 | 3 | 3 | 2020-06-03 | 3 | banana | 1 |
| 3 | 1 | 4 | 1 | 2020-06-15 | 4 | tomato | 0.5 |
| 5 | 2 | 1 | 5 | 2020-06-20 | 1 | apple | 0.50 |
| 6 | 1 | 3 | 3 | 2020-06-20 | 3 | banana | 1 |
| 7 | 5 | 4 | 1 | 2020-06-22 | 4 | tomato | 0.5 |
| 8 | 2 | 3 | 4 | 2020-06-24 | 3 | banana | 1 |
| 9 | 4 | 1 | 2 | 2020-06-25 | 1 | apple | 0.50 |

Vous retrouvez de gauche à droite, d’abord les colonnes de la table *purchases* et ensuite les colonnes de la table *products*. Plusieurs remarques très importantes sur ces résultats:

* la base de données a bien su ajouter les colonnes de *products* en faisant les bonnes correspondances !
* Vous remarquerez que les produits avec les ids 5 et 2, n'apparaissent pas dans les résultats, ce qui est parfaitement logique car personne n’a acheté ces produits dans la table purchases.
* La ligne de *purchases* qui notait la vente d’un produit d’id 32 n’apparaît pas non plus car ce produit n’est pas référencé dans la table *products.*
* Les deux tables ont des noms de colonnes similaires (id), c’est pourquoi l’utilisation des alias permet de lever toute ambiguïté au niveau des résultats.

Remarque: faire un INNER JOIN de A avec B ou de B avec A ne présente aucune différence, le résultat sera le même.

### OUTER JOIN

La jointure de type OUTER JOIN, contrairement à l’INNER JOIN, va garder les lignes pour lesquelles il y a correspondance **ou non**:

| SELECT \* FROM purchases AS pu OUTER JOIN products AS pr ON pu.product\_id = pr.id |
| --- |

Dont les résultats seront:

| **pu.id** | **pu.buyer\_id** | **pu.product\_id** | **pu.quantity** | **pu.date** | **pr.id** | **pr.name** | **pr.price** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 1 | 2 | 2020-06-03 | 1 | apple | 0.50 |
| 2 | 5 | 3 | 3 | 2020-06-03 | 3 | banana | 1 |
| 3 | 1 | 4 | 1 | 2020-06-15 | 4 | tomato | 0.5 |
| 4 | 4 | 32 | 1 | 2020-06-18 | NULL | NULL | NULL |
| 5 | 2 | 1 | 5 | 2020-06-20 | 1 | apple | 0.50 |
| 6 | 1 | 3 | 3 | 2020-06-20 | 3 | banana | 1 |
| 7 | 5 | 4 | 1 | 2020-06-22 | 4 | tomato | 0.5 |
| 8 | 2 | 3 | 4 | 2020-06-24 | 3 | banana | 1 |
| 9 | 4 | 1 | 2 | 2020-06-25 | 1 | apple | 0.50 |
| NULL | NULL | NULL | NULL | NULL | 2 | pear | 1.50 |
| NULL | NULL | NULL | NULL | NULL | 5 | strawberry | 0.1 |

Les résultats se décomposent en trois parties:

* En bleu sont représentées les lignes qui correspondent aux résultats du INNER JOIN
* En vert les lignes de la table *purchases* dont les produits achetés n’ont pas trouvé de correspondance dans la table *products*
* En orange les lignes de *products* qui n’ont jamais été acheté.

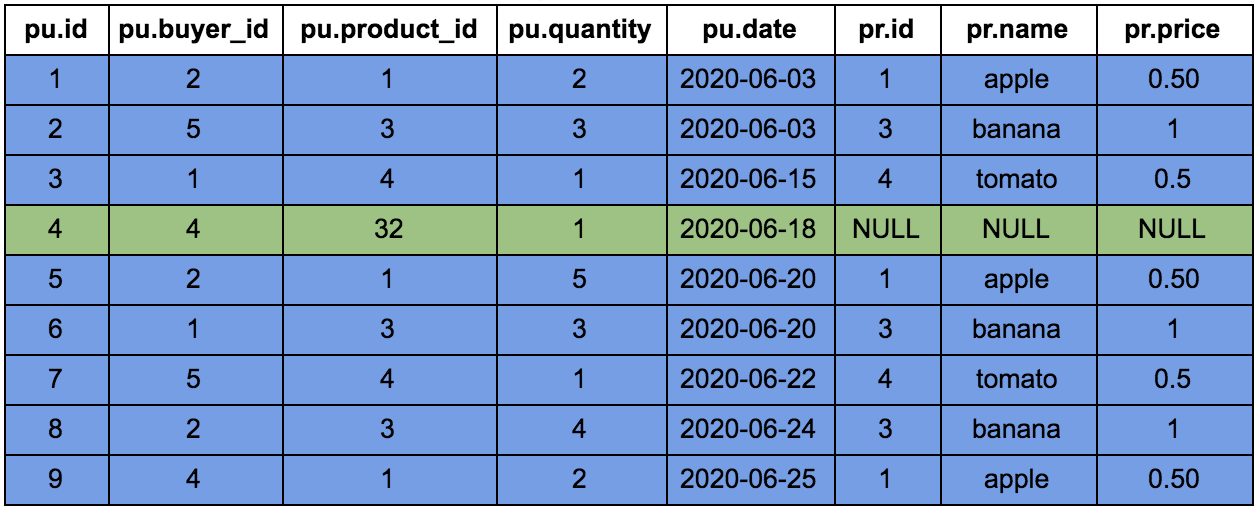
Remarquez bien que toutes les valeurs manquantes ont été automatiquement mises à NULL. De plus faire un OUTER JOIN de A avec B ou de B avec A ne présente aucune différence, le résultat sera le même.

### LEFT JOIN

La clause LEFT JOIN permet de garder toutes les lignes de la table A avec les correspondances de B s’il y en a sinon il mettra des NULL. Elle est la solution intermédiaire entre le INNER JOIN qui ne conserve que les correspondances et le OUTER JOIN qui conserve toutes les lignes des deux tables. Elle est équivalente au RECHERCHEV en Excel.

Reprenons la requête précédente avec un LEFT JOIN et observons les résultats:

| SELECT \* FROM purchases LEFT JOIN products AS pr ON pu.product\_id = pr.id |
| --- |



Vous voyez que nous avons autant de lignes de résultats que de lignes dans la table A *purchases* dont la ligne d’achat du produit 32 qui n’a pas de correspondance dans la table *products*. Cette requête n’est absolument pas équivalente à un LEFT JOIN de la table *products* avec la table *purchases*.

Il existe également une jointure de type RIGHT JOIN qui ne garde que les lignes de la table B avec correspondance avec la table A s’il y en a. Elle est très peu utilisée car elle est équivalente à un LEFT JOIN de B avec A.

On représente souvent les différences de types de jointures avec des graphiques ensemblistes représentant des aires superposées. Elles sont utiles pour garder en tête plus facilement les différentes nature de jointure, vous pouvez observer ces graphiques dans les [annexes](#gfnsea3ax7qy).

## Jointures et opérations

Une opération de jointure renvoie en sortie l’équivalent d’une table contenant des informations des deux tables jointes. On peut voir ça comme une “grande table” en sortie. Vous pouvez ainsi lui appliquer toutes les opérations que nous avons vues ensemble dans ce cours, comme la trier, la grouper par élément ou encore résumer ses informations par des fonctions d’agrégation pour en tirer des informations.

Prenons un exemple simple, on souhaite avoir l’information de la quantité achetée de chaque produit dans le magasin, trié par ordre décroissant. Mais on ne souhaite pas avoir les ids des produits mais bien leurs noms pour avoir des résultats plus intelligibles.

On écrira la requête suivante:

| SELECT pr.name, SUM(pu.quantity) AS total\_quantity FROM purchases AS pu INNER JOIN products AS pr ON pu.product\_id = pr.id GROUP BY 1 ORDER BY 2 DESC |
| --- |

Il faut bien garder en tête que l’ordre des clauses ne changent pas par rapport à ce que l’on a vu, mais que les jointures s’écrivent juste après la clause FROM.

## Multiples jointures

Puisque une jointure produit une “grande” table, une seconde jointure produira une encore plus “grande” table etc… Il est en effet très commun d'utiliser plusieurs jointures en une seule et même requête pour lier les informations de plusieurs tables entre elles. Cela peut paraître très déstabilisant au début mais revenez simplement à l’idée que vous ne faites qu’ajouter finalement des colonnes à votre table de départ.

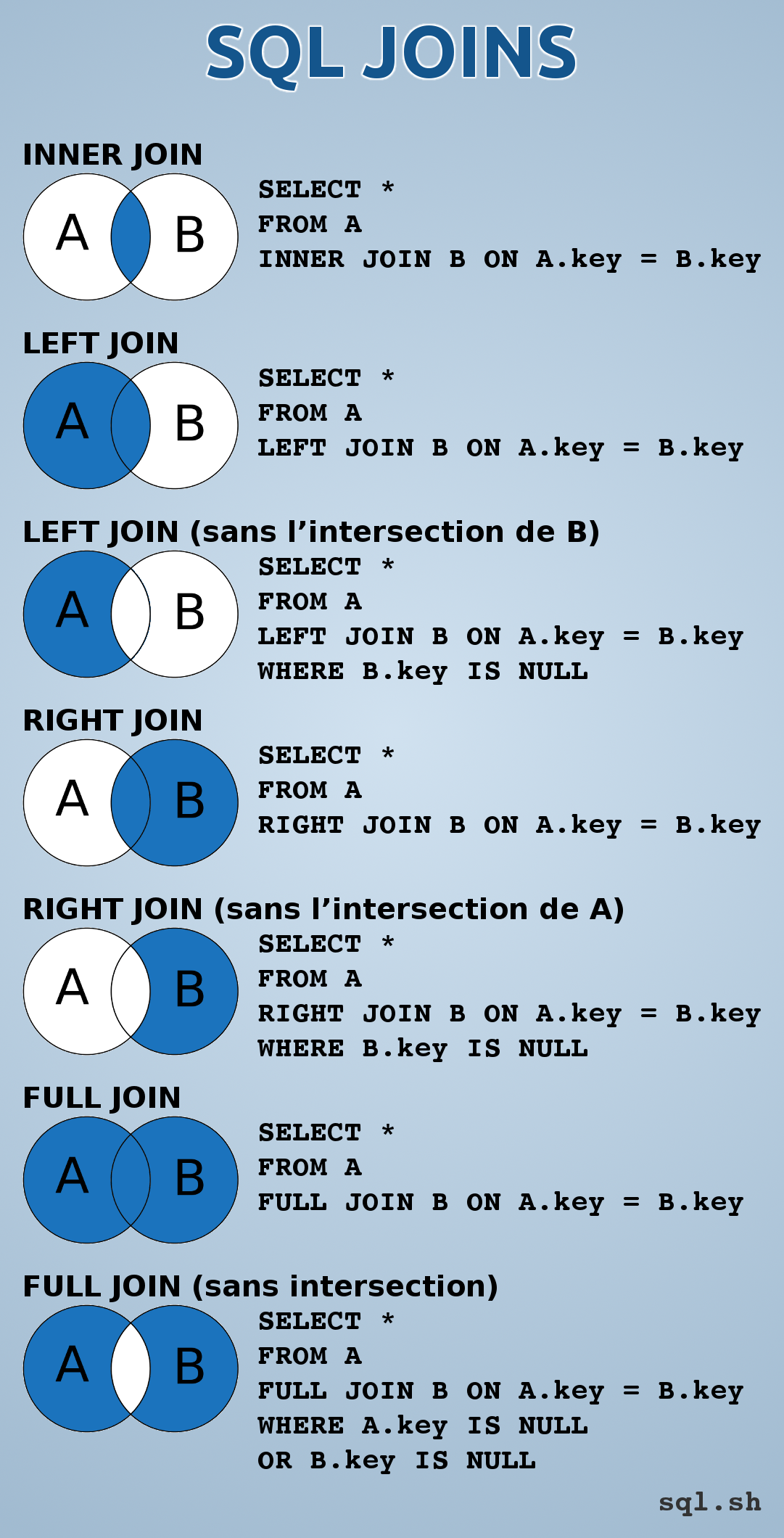
Prenons un exemple d’une requête plus complexe qui cherche à calculer pour chaque acheteur et chaque produit la quantité totale achetée triée par acheteur puis par quantité décroissante.

| SELECT bu.name, pr.name, SUM(pu.quantity) AS total\_quantity FROM purchases AS pu INNER JOIN buyers AS bu ON pu.buyer\_id = bu.id INNER JOIN products AS pr ON pu.product\_id = pr.id GROUP BY 1, 2 ORDER BY 1, 3 DESC |
| --- |

Vous avez maintenant d’excellentes bases SQL qui vous permettront de couvrir un bon ensemble des besoins en entreprise.

# Annexe

## Représentation ensembliste des jointures



# Résumé

* Une table a toujours une clef primaire, c’est-à-dire une colonne qui sert d’identifiant unique pour chacune de ses lignes. Elle peut aussi avoir une ou plusieurs clefs étrangères, c’est-à-dire des colonnes faisant référence aux clefs primaires d’autres tables.
* Par le principe de non redondance d’information du modèle relationnel, il est indispensable de créer des relations entre les tables.
* En SQL, ces relations sont dites des jointures. Une jointure cherche des correspondances entre une table A et une table B via une ou plusieurs colonnes. Généralement ces colonnes sont des clefs étrangères avec des clefs primaires mais ce n’est pas obligatoire. On peut joindre sur n’importe quelle colonne.
* Une jointure peut-être de plusieurs natures mais s’écrit toujours de cette façon:

| FROM TableA AS A TYPE DE JOINTURE TableB AS B ON A.colonne = B.colonne |
| --- |

La jointure s’écrit après la clause FROM et précise sur quelles colonnes joindre avec la clause ON

* Les différents types de jointures sont:
  + INNER JOIN: Elle ne garde que les lignes qui ont bien une correspondance
  + LEFT JOIN: Elle garde toutes les lignes de la table A et vient compléter avec les colonnes de la table B si correspondance il y a, sinon elle mettra des valeurs NULL
  + OUTER JOIN: Elle garde toutes les lignes de A et B avec correspondance ou non
* Les jointures vont ajouter des colonnes à la table de départ produisant ainsi une “grande” table (pour l’image)
* On peut venir faire toutes les opérations usuelles en SQL sur le résultat de cette jointure
* On peut retrouver plusieurs jointures de natures différentes dans une seule et même requête avec différentes tables, autant de fois que nécessaire pour obtenir le résultat recherché.
* Même si les opérations de jointure sont coûteuses d’un point de vue computationnel, le SQL a été pensé pour ces opérations et sont totalement naturelles.
* Si la quantité de jointures est très importante, cela peut vouloir dire que le schéma de la base de données n’est pas optimal.

# Pour aller plus loin

Cours bien illustré pour expliquer les différents types de jointure

<https://sql.sh/cours/jointures>