La requête SELECT

...

Tout ce que vous avez toujours rêvé de savoir sur la requête préférée des français (c'est vendeur, non ?)

Base de données oBlog

Une table *post*
- id : la clé primaire de l'article
- title : le titre de l'article
- summary : son résumé
- publish_date : sa date (et heure) de publication
- author_id : son auteur (ou plutôt l'id de la ligne représentant son auteur dans la table author)
- category_id : sa catégorie (ou plutôt... vous avez compris ?)

Une table *category*
- id : la clé primaire de la catégorie
- title : le titre de la catégorie

Et une table *author*
- id : la clé primaire de l'auteur
- firstname : le prénom de l'auteur
- lastname : son nom de famille

Le SELECT

Un SELECT peut être très simple

```
SELECT * FROM post
```

Ou très complexe

```
SELECT post.title, CONCAT(SUBSTRING(post.content, 1, 50), ' [...]') excerpt
FROM post WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1
MONTH) AND category_id NOT IN (1, 3, 7) ORDER BY publish_date DESC
```

Aïe, pas très lisible, tout ça...

Le SELECT à la loupe

```
SELECT
    post.title,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

Comme ça, c'est déjà mieux, non ?
Ok, commençons donc par le disséquer pour mieux comprendre ce qu'il s'y passe

Le SELECT à la loupe

```
SELECT
    post.title,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

On veut le titre des articles qui seront sélectionnés par cette requête

Le SELECT à la loupe

```
SELECT
    post.title,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

Ainsi que... ouhla... un autre truc...
Ah, on va l'appeler *excerpt* apparemment

Le SELECT à la loupe

```
SELECT
        post.title,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

À y regarder de plus près, on va partir de *content*, qui est une colonne de la table *post*

Le SELECT à la loupe

```
SELECT
        post.title,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

Et on va découper ce que contient la colonne *content* pour ne garder que les 50 premiers caractères

Le SELECT à la loupe

```
SELECT
    post.title,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

Qu'on va concaténer à la chaîne "espace-crochet-trois-petits-points-crochet"... hmm, effectivement, ça va donner un genre d'extrait

Le SELECT à la loupe

```
SELECT
    post.title,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

Bon, le mot-clé SELECT et tout ce qui le suit => OK
FROM, facile, c'est la source de données : la table *post*

Le SELECT à la loupe

```
SELECT
    post.title,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

Chérie ? Ça va filtrer !
Complexe ce filtre... et le AND en dessous me laisse penser qu'il ne va pas être seul en plus...

Le SELECT à la loupe

```
SELECT
    post.title,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

Bon, CURDATE(), c'est la date d'aujourd'hui, merci la doc

Le SELECT à la loupe

```
SELECT
        post.title,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

DATE_SUB() soustrait un intervalle de temps à une date... Donc ça devrait correspondre grosso modo à il y a un mois !

Le SELECT à la loupe

```
SELECT
        post.title,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

Les dates vont crescendo : aujourd'hui est "supérieure" à hier... donc on filtre sur les publications datant d'il y a moins d'un mois !

Le SELECT à la loupe

```
SELECT
    post.title,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

Le deuxième critère a l'air simple mais la syntaxe NOT IN est curieuse

Le SELECT à la loupe

```
SELECT
    post.title,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

Cette syntaxe permet de vérifier si l'opérande de gauche se trouve ou pas dans la liste de droite

Le SELECT à la loupe

```
SELECT
    post.title,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

Donc on exclut les catégories ayant l'id 1, 3 ou 7, voilà tout ! Facile !

Le SELECT à la loupe

```
SELECT
    post.title,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

Et on trie par date de publication décroissante, donc les articles les plus récents en premier

Le SELECT à la loupe

```
SELECT
    post.title,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC
```

Mais au fait, il ne manquerait pas quelque chose à cette requête ?

Le SELECT à la loupe

```
SELECT
	post.title,
	CONCAT(
		SUBSTRING(post.content, 1, 50),
		' [...]'
	) excerpt
FROM post
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC;
```

Ah voilà, avec un point virgule, c'est tout de suite mieux !

Le JOIN

Un bon SELECT se doit d'inclure un des mots-clés les plus cools de la galaxie, celui qui permet d'aller chercher tout un tas de données dans plein de tables en même temps, dans une unique requête : le JOIN ! (prononciation anglaise vivement conseillée)

" Le JOIN, c'est un peu le supermarché de la donnée, ça permet d'aller chercher ta viande, tes légumes et ton bouillon cube pour la cuisine de ce soir, et de racheter des piles LR6 et de l'essuie-tout au passage "

- Un formateur souhaitant rester anonyme

Le SELECT avec un JOIN

```
SELECT
    post.title,
    lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC;
```

Ah elle vous avait manqué ?
La revoilà, la requête complexe, flanquée d'un JOIN pour la rendre plus... complexe !

Le SELECT avec un JOIN

```
SELECT
    post.title,
    lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC;
```

Et, bien entendu, on va rarement faire une jointure pour le plaisir... l'intérêt, c'est de récupérer des données de cette nouvelle table ajoutée à la requête

Le SELECT avec un JOIN

```
SELECT
    post.title,
    lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC;
```

Mais d'ailleurs, pas forcément, on peut aussi faire un JOIN pour filtrer (WHERE) ou trier (ORDER BY) sur un champ de la table jointe

Le SELECT avec un JOIN

```
SELECT
    post.title,
    lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC;
```

Autre remarque : ici, deux écritures de colonnes se côtoient : l'écriture *colonne* et l'écriture *table.colonne*

Le SELECT avec un JOIN

```
SELECT
    post.title,
    lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND category_id NOT IN (1, 3, 7)
ORDER BY publish_date DESC;
```

Les deux sont valides sauf dans le cas d'une ambiguïté : par exemple, si on écrit simplement *id*, car les deux tables ont une colonne *id*

Le SELECT avec un JOIN

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Et en pratique, ce que vous croiserez le plus souvent et que l'on vous encourage à faire, c'est écrire systématique le nom de la table

Le SELECT avec un JOIN

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Si votre collègue jette un oeil à votre requête, il n'aura pas besoin de savoir dans quelle table se trouve quelle colonne, tout est clairement indiqué

Le SELECT vu par un SGBD

Mais alors, comment ça marche un JOIN ? Et pourquoi je ne peux pas utiliser d'alias dans mon filtre WHERE ? Alors que je peux les utiliser dans l'ORDER BY...

Tout simplement parce que l'ordre des mots-clés d'une requête SELECT a été décidé pour qu'il semble le plus naturel possible. Mais le SGBD ne les lit pas pour autant dans cet ordre.

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Déjà, la règle d'or : le SGBD ne vous retourne toujours qu'un seul type de résultat : une table temporaire.

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

La table temporaire est une représentation des résultats correspondant à la requête que vous avez demandée

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Que la table contienne 1 ou 100 colonnes, 3 ou 30 000 lignes, il vous présentera toujours les résultats dans une table temporaire, OK ?

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Et il va commencer par... le FROM. Il va remplir sa table temporaire avec tout ce que contient la table *post*, colonnes et lignes

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Puis, il va chercher d'éventuels JOIN. Pour chaque JOIN trouvé, il va ajouter les colonnes de la table jointe et ...

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Et effectuer sur les données (les lignes) un croisement systématique de chaque ligne issue de sa table temporaire avec chaque ligne de la table jointe

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Si vous avez 150 articles rédigés par 20 auteurs, félicitations, le moteur SQL vient de créer une compilation de 3000 lignes !

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Pas d'inquiétude, le SGBD va simplement préparer le résultat de ce croisement, puis il va regarder le critère de jointure (ON)

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Ce critère va lui permettre de filtrer les lignes incohérentes grâce à un test effectué sur chacune de ces 3000 lignes

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

L'article 8 a été écrit par l'auteur 2, il ne va donc garder que la ligne qui met bout à bout l'article 8 et l'auteur 2 et supprimer les 19 autres combinaisons

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Voilà, le SGBD a constitué une table de 150 lignes contenant toutes les colonnes de *post* + toutes les colonnes d'*author*

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

NB : si la requête contient plusieurs jointures, l'opération est toujours rigoureusement la même, le SGBD va joindre sa table temporaire et la table jointe

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

En croisant toutes les possibilités avant d'éliminer les lignes incohérentes grâce au critère de jointure

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Vient ensuite l'heure de gloire du filtre (WHERE) : le SGBD va prendre toute l'expression située après ce mot-clé (les AND et OR font aussi partie de l'expression)

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Et appliquer ce test à chaque ligne : si le résultat final est faux, la ligne quitte la table temporaire ; s'il est vrai, elle reste. Impitoyable !

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

C’est ensuite (enfin !) au tour du SELECT d’être considéré : écrit en premier, il est en fait traité parmi les derniers

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

D’ailleurs, vous comprenez maintenant pourquoi les alias ne marchent pas dans le filtre (WHERE) ?

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Ils sont déclarés dans le SELECT : au moment où le SGBD interprète le WHERE, il ignore encore comment s'appelleront les données retenues

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Le SELECT, donc. C'est l'heure du grand chamboulement : parfois, on garde tout ; d'autres fois, comme ici, on fait un sacré tri

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Et c'est aussi l'occasion d'exploiter la puissance de calcul des SGBD pour sélectionner des colonnes... *calculées*, tout simplement

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Une colonne calculée, c'est tout simplement une colonne dont le contenu est le résultat d'un calcul sur une ou plusieurs des colonnes de la table temporaire

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Ces colonnes sont ajoutées à la table temporaire après que chaque valeur ait été calculée pour chaque ligne

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Énorme avantage d'avoir filtré d'abord, ces calculs ne seront effectués que sur les lignes que vous souhaitez garder !

Le SELECT vu par un SGBD

```
SELECT
    post.title,
    author.lastname,
    CONCAT(
        SUBSTRING(post.content, 1, 50),
        ' [...]'
    ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Et parce que, si on ne donne pas de nom à ces colonnes calculées, le SGBD les nomme n'importe comment, il est quasi systématique de leur donner un alias

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Et que fait le SGBD des colonnes qui n'ont pas été retenues dans le SELECT ? Il les "grise" : il note sur un coin de nappe qu'elles ne devront pas être affichées mais il les garde

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Car le dernier acteur de la requête entre en scène : le tri. Pourquoi trier en dernier ? Eh bien déjà, rien ne sert de trier des données qui seront filtrées, donc trions après le filtre.

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Et on pourrait vouloir trier sur une colonne calculée, alors trions aussi après le SELECT. Et comme le SELECT arrive en avant-dernier...

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Bref, comme son nom l'indique, le tri trie les données présentes dans la table temporaire. Il est possible de trier sur des alias, puisque le SELECT est déjà passé.

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Il est également possible de trier sur des colonnes non retenues à l'affichage (absentes du SELECT). C'est précisément pour ça que le SGBD n'a rien jeté au SELECT.

Le SELECT vu par un SGBD

```
SELECT
        post.title,
        author.lastname,
        CONCAT(
                SUBSTRING(post.content, 1, 50),
                ' [...]'
        ) excerpt
FROM post
JOIN author ON post.author_id = author.id
WHERE post.publish_date > DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
AND post.category_id NOT IN (1, 3, 7)
ORDER BY post.publish_date DESC;
```

Il est enfin possible de trier sur plusieurs colonnes : pour tout groupe de lignes ayant la même valeur dans la première colonne triée, elles sont ordonnées par la 2e.

Et voilà !

Après de nombreuses étapes et des calculs aussi fastidieux que complexes, le SGBD revient triomphant. Il jette finalement les colonnes qui n'étaient pas retenues au SELECT, elles ne serviront plus à rien. Et il vous présente fièrement le résultat de son travail : vos données !

Et tout ça en quelques dizaines de millisecondes, tout au plus ! Les moteurs SQL sont optimisés pour ce genre de calcul : sur un serveur bien dimensionné, la jointure de 5 tables dont 2 comptent plus de 100 000 lignes, avec des filtres complexes et des colonnes calculées de partout... ne prend que 300 millisecondes. Bon, c'est de la frime : ces résultats peuvent varier en fonction du SGBD, de l'OS (eh oui), de la RAM disponible et des performances du processeur, bien entendu.

Mais si vous demandiez à PHP ou JS (ou C ou Python, peu importe le langage) d'interpréter une requête SQL, c'est-à-dire de bâtir un tableau associatif à partir d'autres tableaux associatifs représentants vos tables puis de filtrer les résultats, de calculer des colonnes supplémentaires et de trier le tout... Eh bien, ça n'aboutirait sûrement même pas. Ou ça mettrait au minimum quelques dizaines de secondes ! Ils ne sont juste pas fait pour ça. Chacun son rôle ;-)