# Module 7 - La suite ? Aller plus loin et SQL en conditions réelles

Quelques notions et conseils pour aller plus loin dans la maîtrise de SQL et dans la conception et l'utilisation des bases relationnelles en situation réelle.

## Écrire des requêtes SQL plus performantes, dans un second temps

1. **Commencer** par écrire **une requête qui marche**
2. L'améliorer **ensuite en inspectant le plan d'exécution** de la requête. Le plan d'exécution de la requête peut être affiché avec l'instruction `EXPLAIN`. Par exemple, `EXPLAIN SELECT Name FROM country WHERE Code Like 'A%';`

## Respecter des conventions

> *Écrivez pour les humains, pas pour la machine.*

- Capitalisation des mots clefs SQL, rester cohérent·e
- Des requêtes bien indentées (aidez-vous de votre IDE)
- Préfixer le nom de chaque colonne par le nom de la table dans vos requêtes `SELECT` lorsque vous réalisez des jointures
- Commentez vos requêtes pour expliquer ce qu'elles font, le *quoi* (pas *comment* elles le font)

## Ne récupérez *que l'essentiel*

> *Conseils aussi utiles pour inspecter une base, apprendre à la connaître*

- limiter le nombre de lignes à remonter avec `LIMIT nb_lignes`
- inspecter le nombre d'enregistrements remontés par votre requête avec `COUNT(*)`
- limiter le nombre de colonnes sur lesquelles requêter, éviter le `SELECT *`
- éviter les calculs non nécessaires
    - `SELECT DISTINCT` est très cher
    - Le *wildcard* `'%'` sur les chaînes de caractères est très cher

## Requêtes imbriquées: sous-requêtes VS *Common Table Expressions* (clause `WITH`)

Les *Common Table Expressions* (CTE) (depuis MySQL v8) permettent d'*aliaser* des requêtes pour les utiliser *comme sous requête*.

S'écrit avec la clause WITH.

```
-- exemple de CTE. Chaque sous-requête est repérée par un alias (cte1, cte2)
-- ce qui permet d'ajouter de la sémantique et des les réutiliser
WITH
  cte1 AS (SELECT a, b FROM table1),
  cte2 AS (SELECT c, d FROM table2)
SELECT b, d FROM cte1 JOIN cte2
WHERE cte1.a = cte2.c;
```

**Les *CTE* sont à privilégier aux sous-requêtes**: plus lisibles, peuvent être réutilisées.

# Diviser un problème complexe en plusieurs problèmes plus simples

Une règle générale c'est *Commencer par réduire, puis combiner*.

```
--l'ordre des opérations est indiqué par le nombre de 1 à 6
SELECT 5
FROM 1
WHERE 2
GROUP BY 3
HAVING 4
ORDER BY 6
```

La jointure a toujours la précédence et *sera donc opérée en premier*. Donc, commencer par filtrer avec une sous-requête ou une Common Table Expressions pour filtrer, **puis** faites les jointures.

# Connaître et tester son moteur de bases de données (SGBDR)

On l'a vu SQL est un standard, et il en existe de nombreuses implémentations (MySQL, MariaDB, PostgreSQL, SQLite, Oracle, MySQL Server, etc.). **Chaque SGBD est une implémentation différente** du standard SQL, et vient avec son *optimisateur de requête*. Une optimisation sur un SGBD ne fonctionnera pas nécessairement sur un autre. **Optimiser demande donc de connaître les détails d'implémentation du SGBD utilisé**.

> *Quand vous faites une recherche (sur Google) pour optimiser une requête, **pensez à indiquer le SGBD utilisé** (MySQL, PostegreSQL, etc.)*

# Bien utiliser les INDEX

Le placement des index dépendra de vos besoins métiers. En général on en placera sur

- les *filtres*. Par exemple si vous filtrez régulièrement vos données sur une colonne, placez-y un index
- les *clefs étrangères*, pour optimiser les jointures.

> *Pour en savoir plus sur l'usage efficace des INDEX, rendez-vous sur ce site.*

# Créer de meilleurs schémas de données (DDL)

Pour obtenir de meilleures performances, il faut commencer par disposer d'une base de données **bien conçue**.

- Normaliser les données: créer autant de tables que nécessaire
- Éviter les colonnes NULLables (et les données NULL)
- Faire des jointures !
- Basée sur l'expérience métier (usage de la base), créer des INDEX pertinents

# Références


- Effective SQL: 61 Specific Ways to Write Better SQL, First Edition, de John L. Viescas, Douglas J. Steele, Ben G. Clothier, publié en 2016 aux éditions O'Reilly. Résumé: *Effective SQL brings together practical solutions and insights so you can solve complex problems with SQL and design databases that simplify data management in the future. It's the only modern book that brings together advanced best practices and realistic example code for all of these versions of SQL: IBM DB2, Microsoft Access, Microsoft SQL Server, MySQL, Oracle Database, and PostgreSQL*
- SQL Pocket Guide, 4th Edition, par Alice Zaho, publié en 2021 aux éditions O'Reilly. Alice Zaho est une data scientist qui a passé plus de 20 ans à travailler sur les bases de données relationnelles. Un livre à destination des data analyst et scientist. Résumé: *If you use SQL in your day-to-day work as a data analyst, data scientist, or data engineer, this popular pocket guide is your ideal on-the-job reference. You'll find many examples that address the language's complexities, along with key aspects of SQL used in Microsoft SQL Server, MySQL, Oracle Database, PostgreSQL, and SQLite.*
- Common Table Expressions (CTE), permettent d'aliaser des requêtes pour les utiliser comme sous requête. S'écrit avec la clause `WITH`.
- Use the index, Luke !, site maintenu par Markus Winand, une référence des bases de données relationnelles. Vous trouverez sur son site une explication complète de l'optimisation et des performances des bases de données relationnelles, notamment via les INDEX. Le contenu est disponible en ligne ou en PDF (traduit en français)