# Exercice TD5 : Java & PostgreSQL (JDBC) - Codons autrement !

Jusqu'à maintenant, nous nous sommes basés essentiellement sur l'approche orientée objet pour effectuer les calculs. Par exemple, lorsque nous voulions obtenir l'état de stock d'un ingrédient, nous avons chargé en tant qu'attribut de la classe Ingrédient, la liste des mouvements de stocks qui lui sont associés et nous avons créé une méthode au sein de la classe même (classe Ingredient), qui permet d'effectuer le calcul de l'état du stock. **L'approche orientée objet reste l'une des meilleures approches dans la programmation** mais elle n'est pas la seule approche possible.

Dans ce TD, nous allons découvrir une approche qui s'appelle le "**database-side processing**" ou encore le "**push-down processing**". Concrètement, cette nouvelle approche se distingue par le fait que les calculs se font directement au niveau de la base de données, ainsi la requête envoyée depuis JDBC ne concerne, généralement, que la récupération des résultats des calculs. Reprenons l'exemple du calcul de stock avec cette approche :

- Avant tout, vous ne devez pas toucher à la méthode existante `StockValue getStockValueAt(Instant t)` dans la classe Ingredient. L'idée est juste d'obtenir le résultat mais à travers une autre approche.

- Les données sont toujours stockées dans la base de données, spécifiquement dans les tables Ingredients et Stock_Movement.

- Pour rappel, l'état de stock fait référence à une date et heure donnée, ce qui oblige à filtrer les mouvements de stocks par une date et heure donnée. Au niveau du SGBD, on peut filtrer les entrées du Stock_movement concerné par l'ajout de conditions à travers la clause "WHERE" dans une requête "SELECT" qui va récupérer le résultat.

- Avec les entrées du Stock_movement filtrées par la date et heure voulue, il faut ensuite faire la somme des quantités des mouvements filtrés, où lorsque le type du mouvement est de type "OUT" (donc sortie) il faut que la valeur devienne négative (par exemple multiplié par -1). Vous pouvez utiliser la syntaxe SQL "CASE … WHEN … ELSE … END" pour substituer le "if … else" au niveau de Java pour pouvoir gérer le cas où le mouvement est de type OUT et qu'il faut rendre la valeur négative avant de faire la somme.

- Les colonnes qui devraient être retournées par la requête SQL doivent être les deux attributs permettant de constituer un objet StockValue qui pour rappel est la quantité de type nombre et l'unité de type unité (L, KG, PCS). La requête donc devrait commencer comme ceci :
  **> select unit, sum(...) as actual_quantity from stock_movement [join ingredient on stock_movement.id_gredient = ingredient.id] where … group by (id_ingredient)**

  N'oubliez pas que **group by** est obligatoire dans la requête SQL car nous calculons l'état de stock par ingrédient, qui au niveau de la base de données est représenté par son identifiant.

La partie verte **[join ingredient on stock_movement.id_gredient = ingredient.id]** est optionnelle dans notre cas car nous n'avons besoin d'aucune colonne de la table Ingredient au niveau de notre résultat.

La partie rouge **...** est à compléter mais doit contenir `CASE … WHEN … ELSE … END` pour pouvoir gérer l'entrée (valeur positive) et sortie (valeur négative) du mouvement de stock.

- Une fois la requête SQL constitué et fonctionnel, dans la classe DataRetriever, il faut créer la méthode `StockValue getStockValueAt(Instant t, Integer ingredientIdentifier)` qui en plus du paramètre spécifiant la date et heure de la valeur du stock voulue, doit prendre également l'identifiant de l'ingrédient voulu, car c'est le seul moyen unique permettant d'identifier un ingrédient au niveau de la base de données. Cette méthode doit intégrer la requête SQL qui permet d'effectuer le calcul, et qui va récupérer le résultat à travers un `ResultSet` pour constituer l'objet `StockValue` attendu.

1) Déroulez l'approche **database-side processing/push-down processing** pour le calcul de l'état de stock d'un ingrédient à un instant donné. Une fois que c'est bon, invoquez la méthode dans la classe Main pour vérifier que le résultat soit le même que celui obtenu à travers le calcul utilisant l'approche orienté objet, en vous assurant à ce que les données utilisées soient identiques.

2) Ajoutez les méthodes suivantes dans la classe DataRetriever, permettant d'obtenir les mêmes résultats avec les mêmes données que l'approche initiale (orientée objet), mais cette fois-ci, en utilisant l'approche **push-down processing** :

    a) **`Doublet getDishCost(Integer dishId)`** : méthode qui calcule le coût des ingrédients d'un plat au niveau de la base de données;

    b) **`Double getGrossMargint(Integer dishId)`** : méthode qui calcule la marge brute d'un plat au niveau de la base de données;

3) Exploitons l'approche **database-side processing** pour optimiser le calcul des statistiques de l'état de stock d'un ingrédient suivant une période donnée. L'idée ici est de savoir l'évolution de l'état de stock sur une périodicité, qui peut être exprimée en jour, en semaine ou en mois. Lors de la récupération, il faut donc à la fois exprimer la périodicité (DAY, WEEK, MONTH) mais surtout une intervalle de date dans laquelle nous allons filtrer les données. Voici un exemple typique de cette fonctionnalité :

    - En entrée : périodicité = JOUR; intervalleMin=01/01/2026; intervalleMax= 05/01/2026

    - Exemple de résultat attendu :

| id_ingredient | 01/01/26 | 02/01/26 | 03/01/26 | 04/01/26 | 05/01/26 |
|---|---|---|---|---|---|
| 1 | 1.0 | 10.5 | 5.5 | 12.5 | 5 |
| 2 | 8.0 | 8.5 | … | … | … |
| 3 | … | … | … | … | … |
| 4 | … | … | … | … | … |
| 5 | … | … | … | … | … |

L'avantage principal d'effectuer le calcul au niveau de la base données pour ce genre de situation (notamment les statistiques) c'est d'éviter de charger beaucoup de données qui peuvent potentiellement surcharger le serveur d'application (Java). L'inconvénient c'est donc que le temps de traitement de la requête peut prendre beaucoup plus de temps, et qu'il faut que le SBGD ait la puissance nécessaire pour effectuer le calcul, si jamais les données sont énormes.

**Implémentez la méthode dans DataRetriever permettant de récupérer le résultat et d'afficher le nom de l'ingrédient avec les statistiques.**