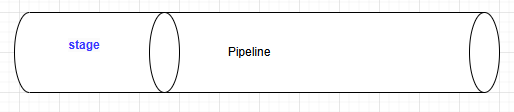
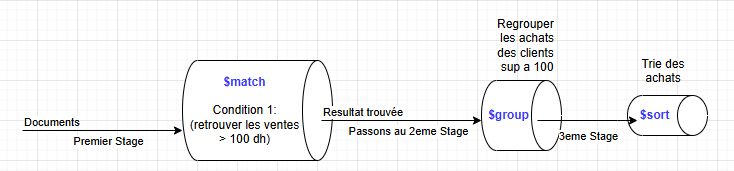
# Introduction aux agrégation pipelines

**1-Qu'est-ce qu'un pipeline ?**

Le pipeline est un processus qui permet de traiter, transformer et analyser les données d'une collection en appliquant une série d'étapes (**stage**) de transformation.



Chaque stage du pipeline prend en entrée les documents de la collection ou les résultats de l’étape précédente, et applique une opération particulière.



Dans l'exemple ci-dessus, le pipeline est optimisé pour restreindre la quantité de données traitées en réduisant la zone de recherche et en minimisant les calculs inutiles pour les étapes suivantes.

**2-Cest quoi la Différence entre le filtrage normal (find) et les pipelines d'agrégations alors ?**

La méthode **find** en MongoDB est principalement utilisée pour récupérer des documents qui correspondent à un critère de recherche spécifique. Elle est limitée lorsqu'il s'agit des calculs complexe ou de transformation c'est la ou les agrégations pipeline vient pour faire.

L’exemple suivant va vous décrire les différences :

Imaginons une collection ventes où chaque document a un champ montant pour indiquer le montant de la vente et un champ categorie pour le type de produit.

Avec **find** , on peut récupérer toutes les ventes d'une certaine catégorie, par exemple :

db.ventes.find({ categorie: "Electronique"})

Cet exemple extrait tous les documents de la catégorie "Electronique" mais ne permet pas de calculer, par exemple, la somme totale des ventes.

Avec un pipeline d'agrégation, en revanche, on peut obtenir le montant total des ventes pour la catégorie "Electronique" :

db.ventes.aggregate([

{$match: { categorie: "Electronique"} }, // Filtre pour ne garder que les documents de la catégorie

{$group: { \_id: "$categorie", totalVentes: { $sum: "$montant" } } } // Calcul de la somme des montants

])

L’agrégation permet donc de faire des calculs plus poussés, ici en calculant le total des ventes, ce que **find** ne peut pas faire directement.

### **1. $match :**

Filtre les documents pour ne garder que ceux qui respectent certaines conditions, similaire à un WHERE en SQL. Souvent, $match est utilisé en début de pipeline pour réduire le volume de données.

Db.vente. Aggregate([{ $match: { statut: "confirmée" } } , ... autres traitements ]);

### **2. $group**

Regroupe les documents par une clé et permet d'appliquer des calculs d'agrégation (comme des sommes, moyennes, etc.) sur chaque groupe.

Db.vente. Aggregate([ match....,{ $group: { \_id: "$categorie", totalVentes: { $sum: "$montant" } } } ] );

### **3. $project**

Sélectionne ou masque certains champs des documents en sortie, et peut également créer des champs calculés, équivalent de select montant , catégorie , montant \* 1.2 as montantAvecTaxe from vente ..... .

Db.vente.Aggregate([{ $project: { montant: 1, categorie: 1, montantAvecTaxe: { $multiply: ["$montant", 1.2] } } }])

### **4. $sort**

Trie les documents selon un ou plusieurs champs, soit en ordre croissant (1) soit en ordre décroissant (-1).

Db.vente.aggregate([{ $sort: { totalVentes: -1 , categorie:1 ...... } }])

### **5. $limit**

Limite le nombre de documents retournés, utile pour paginer les résultats ou pour obtenir les top N résultats.

Db.vente.aggregate([ { $limitt: 5 } ])

### **6. $skip**

Ignore un certain nombre de documents, souvent utilisé avec $**limit** pour paginer les résultats. (sql : Offset )

Db.vente.aggregate([ { $skip: 5 } ])

### **7. $unwind**

Décompose un tableau en plusieurs documents, chacun contenant une valeur du tableau initial. Idéal pour travailler avec des documents qui contiennent des listes.

Db.vente.aggregate([ { $unwind: "$produits" } ])

### **8. $addFields**

Ajoute de nouveaux champs aux documents, souvent pour des calculs ou transformations intermédiaires.

Db.vente.aggregate([ { $addFields: { montantAvecTaxe: { $multiply: ["$montant", 1.2] } } } ])

### **9. $count**

Compte le nombre de documents dans le pipeline. En général, **$count** est utilisé en fin de pipeline pour obtenir le nombre total de documents après les autres étapes de filtrage et de transformation.

Db.vente.aggregate([{ $count: "totalDocuments" }])

### **10. $lookup**

Réalise un "join" (liaison) entre deux collections, similaire à une jointure en SQL. Utilisé pour combiner des documents de différentes collections.

Db.vente.aggregate([{

$lookup: {

from: "clients", // from == le document externe (ou collection)

localField: "clientId", // Champ de clé primaire dans le document local

foreignField: "\_id", // Champ de clé étrangère dans le document externe

as: "detailsClient" // Alias pour créer un tableau contenant les documents correspondants

}

} ])

### **11. $replaceRoot**

Remplace la racine du document courant par un champ spécifique ou un document imbriqué, utile après un **$lookup** pour simplifier les résultats.

{ $replaceRoot: { newRoot: "$detailsClient" } }

### **12. $sample**

Sélectionne aléatoirement un nombre donné de documents. Idéal pour récupérer un échantillon de données.

{ $sample: { size: 5 } }

**4- Documentation et fonctionnalité spécial**

**Vous trouverez ci-dessus les liens pour ce documenter est chercher les méthodes associes à vos besoin**

[Logical Query Operators - MongoDB Manual v8.0](https://www.mongodb.com/docs/manual/reference/operator/query-logical/)Il suffit d’ecrire la methode souhaite et vous trouverez tout un document qui decris la methode et l’utilite de cette derniere.

Sinon pour une recherche rapide ! **QUE POUR ETUDE OU AU CAS DE BESOIN chatGPT** peut être une solution et un **poison SI MAL UTILISER.**

**Merci pour votre attention.**

**(Prochainement un cours sur la migration des schema sql a mongoDB).**