TRAITEMENT DE DONNÉES

Pipelines d'agrégation

Le framework d'agrégation MongoDB pour des traitements complexes

Principes et exemples pratiques — étapes, opérateurs, jointures, facettes, performance et cas d'usage réels

Sommaire

1 Pourquoi le framework d'agrégation ?

Aller au-delà de ce que permet find()

find() sait filtrer, projeter et trier — mais ne sait pas regrouper, calculer des statistiques, joindre des collections ou transformer la forme des documents. Le framework d'agrégation traite les documents à travers une succession d'étapes (stages), chacune recevant le résultat de la précédente.

find()

Filtrer, projeter, trier. Simple et rapide pour des lectures directes.

aggregate()

Filtrer, regrouper, calculer, joindre, transformer — un pipeline complet de traitement.

Analogie

Comme un pipeline Unix : la sortie d'une étape devient l'entrée de la suivante.

2 Anatomie d'un pipeline

Une liste ordonnée d'étapes (stages)

```
db.commandes.aggregate([
  { $match: { statut: "livree" } },        // étape 1
  { $group: { _id: "$client",              // étape 2
      total: { $sum: "$montant" } } },
  { $sort: { total: -1 } },                 // étape 3
  { $limit: 10 }                            // étape 4
])
```

Chaque étape reçoit les documents produits par l'étape précédente — jamais la collection d'origine, sauf à la toute première étape.

3 $match — filtrer les documents

Le même langage de filtre que find()

```
db.commandes.aggregate([
  { $match: { statut: "livree", montant: { $gt: 50 } } }
])
```

Pourquoi le placer tôt

Un $match placé en première étape réduit immédiatement le nombre de documents traités par les étapes suivantes — et peut exploiter un index, exactement comme un find().

Peut apparaître plusieurs fois

Rien n'empêche un second $match plus loin dans le pipeline, par exemple après un $group, pour filtrer sur un résultat agrégé (équivalent d'un HAVING en SQL).

4 $project — remodeler et calculer

Choisir, renommer ou calculer des champs

```
db.commandes.aggregate([
  { $project: {
      client: 1,
      annee: { $year: "$date" },
      total_ttc: { $multiply: ["$montant", 1.2] },
      _id: 0
  } }
])
```

Contrairement à la projection de find() (0/1 uniquement), $project peut aussi calculer de nouveaux champs à partir des champs existants.

5 $group — regrouper et agréger

L'étape la plus utilisée pour les statistiques

```
db.commandes.aggregate([
  { $group: {
      _id: "$client",
      total_depense: { $sum: "$montant" },
      nombre_commandes: { $sum: 1 }
  } }
])
```

Le champ _id

_id définit la clé de regroupement : un document en sortie par valeur distincte.

_id: null regroupe tous les documents en un seul résultat global (total général, par exemple).

Cas d'usage : chiffre d'affaires par client, nombre de commandes par jour, moyenne de note par produit.

6 Les accumulateurs courants

Utilisables dans $group (et $bucket, $setWindowFields)

Accumulateur	Rôle
`$sum`	Somme des valeurs (ou compte les documents avec $sum: 1)
`$avg`	Moyenne des valeurs
`$min / $max`	Valeur minimale / maximale
`$push`	Empile les valeurs dans un tableau (avec doublons)
`$addToSet`	Empile les valeurs uniques dans un tableau
`$first / $last`	Première / dernière valeur rencontrée (utile après un $sort)

$push et $addToSet sont pratiques pour reconstituer des listes — ex. les produits achetés par client dans un même document de résultat.

7 $sort, $limit, $skip

Trier et paginer un résultat de pipeline

```
db.commandes.aggregate([
  { $group: { _id: "$client", total: { $sum: "$montant" } } },
  { $sort: { total: -1 } },    // du plus gros dépensier au plus petit
  { $skip: 10 },               // page 2 (10 par page)
  { $limit: 10 }
])
```

Ordre des étapes important

$sort avant $limit donne le top N ; $limit avant $sort limiterait arbitrairement avant de trier — un piège fréquent. MongoDB peut optimiser un $sort suivi d'un $limit en interne (top-k sort) pour éviter de trier l'ensemble du jeu de données.

8 $unwind — déplier les tableaux

Transformer un tableau en plusieurs documents

```
// commande = { client: "Marie", articles: [
//   { produit: "Clavier", quantite: 1 },
//   { produit: "Souris", quantite: 2 } ] }

db.commandes.aggregate([
  { $unwind: "$articles" },
  { $group: { _id: "$articles.produit",
      total_vendu: { $sum: "$articles.quantite" } } }
])
```

Chaque commande à N articles devient N documents (un par article) avant le $group — indispensable pour agréger au niveau de l'article plutôt que de la commande.

9 $lookup — jointures entre collections

L'équivalent d'un LEFT JOIN en SQL

```
db.commandes.aggregate([
  { $lookup: {
      from: "clients",
      localField: "client_id",
      foreignField: "_id",
      as: "infos_client"
  } }
])
```

Résultat

Chaque commande reçoit un nouveau champ infos_client, un tableau contenant le(s) document(s) correspondant(s) de la collection clients.

Combiner avec $unwind sur infos_client si on attend une seule correspondance (relation 1-1).

Cas d'usage : enrichir des commandes avec les infos client, des articles avec leur catégorie complète, etc.

10 $lookup avancé (sous-pipeline)

Une jointure avec ses propres conditions et étapes

```
db.clients.aggregate([
  { $lookup: {
      from: "commandes",
      let: { id_client: "$_id" },
      pipeline: [
        { $match: { $expr: {
            $and: [ { $eq: ["$client_id", "$$id_client"] },
                    { $eq: ["$statut", "livree"] } ] } } },
        { $count: "nombre" }
      ],
      as: "commandes_livrees"
  } }
])
```

11 $facet — plusieurs analyses en parallèle

Calculer plusieurs résultats en un seul aller-retour

```
db.produits.aggregate([
  { $facet: {
      par_categorie: [ { $group: { _id: "$categorie", n: { $sum: 1 } } } ],
      prix_extremes: [ { $group: { _id: null,
          min: { $min: "$prix" }, max: { $max: "$prix" } } } ],
      total: [ { $count: "nombre" } ]
  } }
])
```

Un seul appel produit trois analyses différentes de la même collection — utile pour construire un tableau de bord sans multiplier les requêtes.

12 $bucket — histogrammes de répartition

Répartir des documents en tranches (buckets)

```
db.produits.aggregate([
  { $bucket: {
      groupBy: "$prix",
      boundaries: [0, 25, 50, 100, 500],
      default: "500+",
      output: { nombre: { $sum: 1 } }
  } }
])
```

Répartit les produits en tranches de prix (0-25, 25-50, 50-100, 100-500, 500+) avec le nombre de produits par tranche — la base d'un histogramme.

13 Opérateurs d'expression

Calculer des valeurs à l'intérieur d'un stage

Opérateur	Rôle
`$cond`	Condition if/then/else en une expression
`$switch`	Plusieurs branches conditionnelles (équivalent switch/case)
`$year, $month, $dayOfWeek`	Extraction de composantes d'une date
`$concat`	Concaténer des chaînes de caractères
`$size`	Taille d'un tableau

```
{ $project: { tranche_age: { $cond: {
    if: { $gte: ["$age", 18] }, then: "adulte", else: "mineur"
} } } }
```

14 Variables $$ROOT et $$this

Référencer le document courant dans une expression

$$ROOT

Représente le document entier en cours de traitement — utile pour le conserver intact tout en calculant un regroupement.

$$this

Représente l'élément courant à l'intérieur d'une expression itérant sur un tableau (ex. $map, $filter).

```
{ $group: { _id: "$client", derniere_commande: { $last: "$$ROOT" } } }
```

15 Performance des pipelines

Un pipeline mal ordonné coûte cher

Placer $match et $sort tôt

Réduit le volume de documents traités par les étapes suivantes, et peut exploiter un index.

Indexer les champs de $match / $sort initiaux

Comme pour find() — sans index, ces étapes déclenchent un COLLSCAN.

$project après $match

Ne pas remodeler des documents qui seront filtrés juste après — inverser l'ordre gaspille du travail.

explain() sur un pipeline

db.collection.aggregate([...], { explain: true }) — ou .explain() — révèle le plan réellement exécuté.

16 Écrire le résultat ($out, $merge)

Matérialiser un pipeline dans une collection

```
// remplace entièrement la collection cible
db.commandes.aggregate([
  { $group: { _id: "$client", total: { $sum: "$montant" } } },
  { $out: "resume_clients" }
])

// fusionne avec une collection existante (upsert par clé)
db.commandes.aggregate([
  { $group: { ... } },
  { $merge: { into: "resume_clients", whenMatched: "replace" } }
])
```

17 Cas pratique — tableau de bord des ventes

Combiner plusieurs stages pour un rapport complet

```
db.commandes.aggregate([
  { $match: { statut: "livree", date: { $gte: ISODate("2026-01-01") } } },
  { $unwind: "$articles" },
  { $group: {
      _id: "$articles.produit",
      quantite_totale: { $sum: "$articles.quantite" },
      chiffre_affaires: { $sum: { $multiply: ["$articles.quantite", "$articles.prix_unitaire"] } }
  } },
  { $sort: { chiffre_affaires: -1 } },
  { $limit: 5 }
])
```

18 Cas pratique — rapport multi-étapes

$lookup + $unwind + $group + $facet combinés

Objectif : pour chaque client, obtenir son nombre de commandes livrées, son montant total dépensé, et en parallèle une répartition globale des clients par tranche de dépense.

1	`$match`	Ne garder que les commandes livrées
2	`$group`	Regrouper par client : total dépensé, nombre de commandes
3	`$lookup`	Enrichir avec le nom et l'email du client
4	`$facet`	En parallèle : liste détaillée + répartition par tranche ($bucket)

Ce type de pipeline remplace ce qui exigerait plusieurs requêtes séparées et un assemblage manuel côté application.

19 Bonnes pratiques et pièges fréquents

Ce qui revient le plus souvent en revue de code

$unwind sur un gros tableau

Multiplie le nombre de documents traités — vérifier l'impact sur les étapes suivantes.

$lookup sans index sur foreignField

Transforme la jointure en parcours complet de la collection cible à chaque document.

Pipeline trop long sans $match intermédiaire

Traiter des documents inutiles à chaque étape coûte cher — filtrer dès que possible.

Oublier allowDiskUse pour un gros $group/$sort

Au-delà de 100 Mo en mémoire, une étape échoue sans cette option activée.

Pour aller plus loin

- **Documentation officielle** — mongodb.com/docs/manual/aggregation
- **Référence des opérateurs** — Liste complète des stages et expressions disponibles
- **MongoDB Compass** — Générateur visuel de pipeline avec aperçu à chaque étape
- **Pour continuer** — Un TP dédié : construire pas à pas le tableau de bord du cas pratique

Merci — un pipeline se construit étape par étape, en vérifiant le résultat à chaque ajout.