Formation rBDD - NoSQL-Mongo-YS

TP — S'approprier les pipelines d'agrégation MongoDB

Prérequis : bases CRUD MongoDB (voir le TP-1 Mongo). Durée indicative : 1h30 à 2h.

Objectif : manipuler concrètement chaque étape (stage) du framework d'agrégation vue dans le cours — $match, $group, $sort, $unwind, $lookup, $facet, $bucket — sur un jeu de données réaliste de boutique en ligne, jusqu'à construire un tableau de bord complet.

Étape 0 — Préparer les données

Démarrer MongoDB et se connecter :

```
docker run -d --name mon-mongo -p 27017:27017 mongo
docker exec -it mon-mongo mongosh
```

Créer la base et insérer un jeu de données de départ :

```
use boutique
```

```
db.clients.insertMany([
  { _id: 1, nom: "Marie Dupont", email: "marie@mail.fr", ville: "Orléans" },
  { _id: 2, nom: "Lucas Martin", email: "lucas@mail.fr", ville: "Paris" },
  { _id: 3, nom: "Sofia Bernard", email: "sofia@mail.fr", ville: "Lyon" }
])

db.commandes.insertMany([
  {
    client_id: 1, statut: "livree", date: ISODate("2026-01-15"),
    articles: [
      { produit: "Clavier", quantite: 1, prix_unitaire: 79.90 },
      { produit: "Souris", quantite: 2, prix_unitaire: 24.90 }
    ]
  },
  {
    client_id: 1, statut: "livree", date: ISODate("2026-02-03"),
    articles: [ { produit: "Écran", quantite: 1, prix_unitaire: 249.00 } ]
  },
  {
    client_id: 2, statut: "livree", date: ISODate("2026-01-20"),
    articles: [
      { produit: "Clavier", quantite: 1, prix_unitaire: 79.90 },
      { produit: "Webcam", quantite: 1, prix_unitaire: 39.90 }
    ]
  },
  {
    client_id: 2, statut: "annulee", date: ISODate("2026-02-10"),
    articles: [ { produit: "Casque", quantite: 1, prix_unitaire: 59.90 } ]
  },
  {
    client_id: 3, statut: "livree", date: ISODate("2026-02-18"),
    articles: [
      { produit: "Souris", quantite: 1, prix_unitaire: 24.90 },
```

```
      { produit: "Casque", quantite: 1, prix_unitaire: 59.90 },
      { produit: "Écran", quantite: 2, prix_unitaire: 249.00 }
    ]
  }
])
```

Vérifier :

```
db.clients.countDocuments()    // 3
db.commandes.countDocuments()  // 5
```

Étape 1 — $match : filtrer comme avec find()

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

Un pipeline d'une seule étape se comporte comme un find() classique — c'est le point de départ naturel de tout pipeline.

Exercice 1 : écrivez un pipeline qui ne garde que les commandes livrées après le 1er février 2026 ($gte).

Étape 2 — $project : remodeler les documents

```
db.commandes.aggregate([
  { $project: {
      client_id: 1,
      annee: { $year: "$date" },
      mois: { $month: "$date" },
      nombre_articles: { $size: "$articles" },
      _id: 0
  } }
])
```

$size calcule la taille d'un tableau — ici, le nombre de lignes d'articles par commande (pas la quantité totale, voir étape 4).

Exercice 2 : ajoutez un champ est_recente qui vaut true si la commande date d'après le 1er février 2026, false sinon, en utilisant $cond.

Étape 3 — $group : premières statistiques

Chiffre d'affaires total (commandes livrées uniquement) — attention, à ce stade montant n'existe pas encore par commande puisque le prix est dans le tableau articles ; on additionne donc via une expression intermédiaire :

```
db.commandes.aggregate([
  { $match: { statut: "livree" } },
  { $project: {
      client_id: 1,
```

```
      montant_commande: {
        $sum: {
          $map: {
            input: "$articles",
            as: "a",
            in: { $multiply: ["$$a.quantite", "$$a.prix_unitaire"] }
          }
        }
      }
  } },
  { $group: {
      _id: "$client_id",
      total_depense: { $sum: "$montant_commande" },
      nombre_commandes: { $sum: 1 }
  } },
  { $sort: { total_depense: -1 } }
])
```

Cette étape introduit $map (calcule le montant de chaque commande en sommant articles) avant de regrouper par client — une première illustration de pipeline à plusieurs étapes qui se préparent l'une l'autre.

Exercice 3 : modifiez le pipeline pour que _id vaille null et obtenez le chiffre d'affaires total tous clients confondus en un seul document.

Étape 4 — $unwind : déplier les articles

L'approche avec $map ci-dessus fonctionne, mais dès qu'on veut agréger par produit (et non par commande), il faut déplier le tableau articles en documents séparés :

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

Comparez mentalement avec l'étape 3 : ici chaque article individuel devient la clé de regroupement, plutôt que la commande entière.

Exercice 4 : à partir de ce pipeline, trouvez le produit le plus vendu en quantité (pas en chiffre d'affaires) — ajoutez $limit: 1 après un tri adapté.

Étape 5 — $lookup : enrichir avec les infos client

```
db.commandes.aggregate([
  { $match: { statut: "livree" } },
  { $lookup: {
      from: "clients",
```

```
      localField: "client_id",
      foreignField: "_id",
      as: "infos_client"
  } },
  { $unwind: "$infos_client" },
  { $project: {
      _id: 0,
      client: "$infos_client.nom",
      ville: "$infos_client.ville",
      date: 1
  } }
])
```

$unwind juste après $lookup est un motif très courant : il transforme le tableau infos_client (toujours à un seul élément ici, car client_id est unique) en un objet simple, plus pratique pour le $project qui suit.

Exercice 5 : écrivez un pipeline qui affiche, pour chaque commande livrée, le nom du client et la ville, filtré sur la seule ville "Orléans".

Étape 6 — $facet : plusieurs analyses en un seul appel

```
db.commandes.aggregate([
  { $match: { statut: "livree" } },
  { $facet: {
      par_ville: [
        { $lookup: { from: "clients", localField: "client_id", foreignField:
"_id", as: "c" } },
        { $unwind: "$c" },
        { $group: { _id: "$c.ville", nombre_commandes: { $sum: 1 } } }
      ],
      top_produits: [
        { $unwind: "$articles" },
        { $group: { _id: "$articles.produit", quantite: { $sum:
"$articles.quantite" } } },
        { $sort: { quantite: -1 } },
        { $limit: 3 }
      ],
      total_commandes: [ { $count: "nombre" } ]
  } }
])
```

Un seul appel produit trois analyses indépendantes de la même collection filtrée — chaque branche de $facet est elle-même un mini-pipeline complet.

Exercice 6 : ajoutez une quatrième branche chiffre_affaires_global qui calcule le total général (utilisez $unwind + $group avec _id: null).

Étape 7 — $bucket : répartition par tranche

```
db.commandes.aggregate([
  { $match: { statut: "livree" } },
  { $unwind: "$articles" },
  { $bucket: {
      groupBy: "$articles.prix_unitaire",
```

```
      boundaries: [0, 30, 80, 300],
      default: "300+",
      output: { nombre_articles: { $sum: 1 } }
  } }
])
```

Répartit les lignes d'articles vendus par tranche de prix unitaire — la base d'un histogramme de gamme de prix.

Exercice 7 : changez les tranches (boundaries) pour obtenir une répartition plus fine (par exemple 0, 25, 50, 100, 300), et observez comment default capte tout ce qui dépasse la dernière borne.

Étape 8 — Vérifier la performance

```
db.commandes.createInde{ statut: 1 })
```

```
db.commandes.aggregate([
  { $match: { statut: "livree" } },
  { $group: { _id: "$client_id", total: { $sum: 1 } } }
]).explain("executionStats")
```

Dans la sortie, cherchez stage: "IXSCAN" au niveau du $match — signe que l'index est utilisé plutôt qu'un parcours complet (COLLSCAN).

Exercice 8 : créez l'index, relancez explain(), et vérifiez la différence par rapport à l'exécution sans index (vous pouvez le supprimer avec db.commandes.dropInde{ statut: 1 }) pour comparer).

Étape 9 — Synthèse : le tableau de bord complet

Reprenez les étapes précédentes pour construire, en un seul pipeline, un tableau de bord qui donne pour chaque client : son nombre de commandes livrées, son montant total dépensé, et son produit le plus acheté (en quantité).

Exercice 9 (synthèse) : construisez ce pipeline vous-même avant de regarder le corrigé. Indices : $match → $unwind → $group (deux niveaux, ou un $group avec $push suivi d'un traitement) → $lookup pour le nom du client → $sort.

Corrigés

Corrigé Exercice 1

```
db.commandes.aggregate([
  { $match: { statut: "livree", date: { $gte: ISODate("2026-02-01") } } }
])
```

Corrigé Exercice 2

```
db.commandes.aggregate([
  { $project: {
      client_id: 1,
      est_recente: { $cond: { if: { $gte: ["$date", ISODate("2026-02-01")] },
then: true, else: false } }
  } }
])
```

Corrigé Exercice 3

```
db.commandes.aggregate([
  { $match: { statut: "livree" } },
  { $project: {
      montant_commande: {
        $sum: { $map: { input: "$articles", as: "a", in: { $multiply: ["$
$a.quantite", "$$a.prix_unitaire"] } } }
      }
  } },
  { $group: { _id: null, chiffre_affaires_total: { $sum: "$montant_commande" } }
}
])
```

Corrigé Exercice 4

```
db.commandes.aggregate([
  { $match: { statut: "livree" } },
  { $unwind: "$articles" },
  { $group: { _id: "$articles.produit", quantite_totale: { $sum:
"$articles.quantite" } } },
  { $sort: { quantite_totale: -1 } },
  { $limit: 1 }
])
```

Corrigé Exercice 5

```
db.commandes.aggregate([
  { $match: { statut: "livree" } },
  { $lookup: { from: "clients", localField: "client_id", foreignField: "_id",
as: "infos_client" } },
  { $unwind: "$infos_client" },
  { $match: { "infos_client.ville": "Orléans" } },
  { $project: { _id: 0, client: "$infos_client.nom", ville:
"$infos_client.ville" } }
])
```

Corrigé Exercice 6

```
db.commandes.aggregate([
  { $match: { statut: "livree" } },
  { $facet: {
      par_ville: [
        { $lookup: { from: "clients", localField: "client_id", foreignField:
"_id", as: "c" } },
        { $unwind: "$c" },
        { $group: { _id: "$c.ville", nombre_commandes: { $sum: 1 } } }
      ],
      top_produits: [
        { $unwind: "$articles" },
```

```
        { $group: { _id: "$articles.produit", quantite: { $sum:
"$articles.quantite" } } },
        { $sort: { quantite: -1 } },
        { $limit: 3 }
      ],
      total_commandes: [ { $count: "nombre" } ],
      chiffre_affaires_global: [
        { $unwind: "$articles" },
        { $group: { _id: null, total: { $sum: { $multiply:
["$articles.quantite", "$articles.prix_unitaire"] } } } }
      ]
  } }
])
```

Corrigé Exercice 7

```
db.commandes.aggregate([
  { $match: { statut: "livree" } },
  { $unwind: "$articles" },
  { $bucket: {
      groupBy: "$articles.prix_unitaire",
      boundaries: [0, 25, 50, 100, 300],
      default: "300+",
      output: { nombre_articles: { $sum: 1 } }
  } }
])
```

Corrigé Exercice 8

```
// sans index
db.commandes.dropInde{ statut: 1 })
db.commandes.aggregate([{ $match: { statut:
"livree" } }]).explain("executionStats")
// → stage: "COLLSCAN" attendu

// avec index
db.commandes.createInde{ statut: 1 })
db.commandes.aggregate([{ $match: { statut:
"livree" } }]).explain("executionStats")
// → stage: "IXSCAN" attendu
```

Corrigé Exercice 9 (synthèse)

```
db.commandes.aggregate([
  { $match: { statut: "livree" } },
  { $unwind: "$articles" },
  { $group: {
      _id: { client_id: "$client_id", produit: "$articles.produit" },
      quantite_produit: { $sum: "$articles.quantite" },
      montant_produit: { $sum: { $multiply: ["$articles.quantite",
"$articles.prix_unitaire"] } }
  } },
  { $sort: { quantite_produit: -1 } },
  { $group: {
      _id: "$_id.client_id",
      total_depense: { $sum: "$montant_produit" },
      nombre_lignes_articles: { $sum: 1 },
      produit_prefere: { $first: "$_id.produit" }
  } },
  { $lookup: { from: "clients", localField: "_id", foreignField: "_id", as:
"infos" } },
```

```
  { $unwind: "$infos" },
  { $project: {
      _id: 0,
      client: "$infos.nom",
      total_depense: 1,
      produit_prefere: 1
  } },
  { $sort: { total_depense: -1 } }
])
```

Le double $group est la clé : le premier calcule la quantité par (client, produit), le tri place ensuite le produit le plus acheté en tête pour chaque client, et le second $group récupère ce premier produit via $first tout en sommant le total dépensé.

Pour la suite

Une fois ces pipelines maîtrisés, deux directions naturelles : écrire le résultat d'un pipeline dans une collection avec $out ou $merge (module 16 du cours) pour matérialiser ce tableau de bord et le requêter directement, ou explorer $setWindowFields pour des calculs de fenêtre glissante (moyenne mobile, rang) qui vont au-delà de ce TP.