**Aggregate**La fonction aggregate de MongoDB est utilisée pour effectuer des opérations complexes de traitement de données, comme le filtrage, le tri, le regroupement, et la transformation de documents dans une collection. Elle est souvent utilisée pour analyser et résumer des données.

Voici un guide simple pour débuter avec aggregate en MongoDB, avec des exemples.

**1. Syntaxe de base**

La fonction aggregate prend un tableau de stages (ou "étapes") qui définissent les opérations à appliquer aux documents de la collection. Chaque stage est représenté par un objet qui utilise un opérateur d'agrégation, comme $match, $group, $sort, etc.

javascript

Copier le code

db.collection.aggregate([

{ /\* étape 1 \*/ },

{ /\* étape 2 \*/ },

// ...

])

**2. Exemples d'utilisation**

**Exemple de données**

Imaginons une collection students contenant les documents suivants :

json

Copier le code

{

"\_id": 1,

"name": "Alice",

"age": 20,

"score": 85,

"class": "A"

}

{

"\_id": 2,

"name": "Bob",

"age": 22,

"score": 90,

"class": "B"

}

{

"\_id": 3,

"name": "Charlie",

"age": 23,

"score": 75,

"class": "A"

}

{

"\_id": 4,

"name": "David",

"age": 21,

"score": 80,

"class": "B"

}

**3. Les opérations les plus courantes**

**3.1 Filtrage avec $match**

L'opérateur $match est utilisé pour filtrer les documents en fonction de certaines conditions, semblable à une requête find.

javascript

Copier le code

db.students.aggregate([

{ $match: { class: "A" } }

])

Cela retourne tous les étudiants de la classe "A".

**3.2 Regroupement avec $group**

L'opérateur $group permet de regrouper les documents et d'effectuer des calculs agrégés comme la somme ($sum), la moyenne ($avg), etc.

javascript

Copier le code

db.students.aggregate([

{ $group: { \_id: "$class", averageScore: { $avg: "$score" } } }

])

Cela calcule la moyenne des scores par classe.

**Résultat :**

json

Copier le code

{ "\_id": "A", "averageScore": 80 }

{ "\_id": "B", "averageScore": 85 }

**3.3 Tri avec $sort**

L'opérateur $sort permet de trier les documents en fonction d'un champ.

javascript

Copier le code

db.students.aggregate([

{ $sort: { score: -1 } }

])

Cela trie les étudiants par score en ordre décroissant.

**3.4 Limiter le nombre de documents avec $limit**

L'opérateur $limit limite le nombre de documents retournés.

javascript

Copier le code

db.students.aggregate([

{ $sort: { score: -1 } },

{ $limit: 2 }

])

Cela retourne les deux étudiants avec les scores les plus élevés.

**3.5 Projet avec $project**

L'opérateur $project permet de sélectionner ou transformer les champs à afficher.

javascript

Copier le code

db.students.aggregate([

{ $project: { name: 1, score: 1, \_id: 0 } }

])

Cela ne retourne que les champs name et score pour chaque document, sans afficher l’\_id.

**3.6 Calculer des statistiques avec $group et $count**

Vous pouvez compter le nombre de documents en utilisant $count.

javascript

Copier le code

db.students.aggregate([

{ $match: { class: "A" } },

{ $count: "total\_students" }

])

Cela retourne le nombre total d’étudiants dans la classe "A".

**4. Combiner plusieurs opérations**

Vous pouvez enchaîner plusieurs stages pour créer des pipelines plus complexes. Par exemple, filtrons d'abord les étudiants ayant un score de plus de 80, puis calculons la moyenne des scores par classe.

javascript

Copier le code

db.students.aggregate([

{ $match: { score: { $gt: 80 } } },

{ $group: { \_id: "$class", averageScore: { $avg: "$score" } } }

])

Cela filtre les étudiants avec un score supérieur à 80, puis calcule la moyenne des scores pour chaque classe ayant des étudiants répondant à ce critère.

**5. Exemple avancé avec plusieurs étapes**

Disons que nous voulons trouver la moyenne des scores par classe pour les étudiants âgés de plus de 21 ans, puis trier les classes en fonction de la moyenne des scores en ordre décroissant.

javascript

Copier le code

db.students.aggregate([

{ $match: { age: { $gt: 21 } } },

{ $group: { \_id: "$class", averageScore: { $avg: "$score" } } },

{ $sort: { averageScore: -1 } }

])

L'opérateur $unwind en MongoDB est utilisé pour "détacher" ou "déplier" des documents contenant des tableaux. Si un champ d'un document est un tableau, $unwind va créer un document distinct pour chaque élément de ce tableau. Cela permet de traiter chaque élément individuellement dans le pipeline d'agrégation.

**Exemple de base avec $unwind**

Supposons que nous avons une collection orders qui contient des commandes, et chaque commande a une liste de produits achetés.

json

Copier le code

{

"\_id": 1,

"customer": "Alice",

"items": [

{ "product": "Laptop", "quantity": 1 },

{ "product": "Mouse", "quantity": 2 }

]

}

{

"\_id": 2,

"customer": "Bob",

"items": [

{ "product": "Keyboard", "quantity": 1 },

{ "product": "Monitor", "quantity": 1 },

{ "product": "Mouse", "quantity": 1 }

]

}

Avec $unwind, on peut "détacher" chaque élément du tableau items pour créer un document par produit acheté.

**Utilisation de $unwind**

Voici comment utiliser $unwind pour obtenir chaque produit de chaque commande comme un document individuel :

javascript

Copier le code

db.orders.aggregate([

{ $unwind: "$items" }

])

Cela produit le résultat suivant :

json

Copier le code

{ "\_id": 1, "customer": "Alice", "items": { "product": "Laptop", "quantity": 1 } }

{ "\_id": 1, "customer": "Alice", "items": { "product": "Mouse", "quantity": 2 } }

{ "\_id": 2, "customer": "Bob", "items": { "product": "Keyboard", "quantity": 1 } }

{ "\_id": 2, "customer": "Bob", "items": { "product": "Monitor", "quantity": 1 } }

{ "\_id": 2, "customer": "Bob", "items": { "product": "Mouse", "quantity": 1 } }

Chaque élément de items devient un document séparé avec le même \_id et customer que l'original.

**Exemple pratique : Calcul du total des quantités par produit**

Disons qu'on veut savoir combien de chaque produit a été commandé au total.

javascript

Copier le code

db.orders.aggregate([

{ $unwind: "$items" },

{ $group: { \_id: "$items.product", totalQuantity: { $sum: "$items.quantity" } } }

])

**Résultat :**

json

Copier le code

{ "\_id": "Laptop", "totalQuantity": 1 }

{ "\_id": "Mouse", "totalQuantity": 3 }

{ "\_id": "Keyboard", "totalQuantity": 1 }

{ "\_id": "Monitor", "totalQuantity": 1 }

**Options supplémentaires de $unwind**

1. **preserveNullAndEmptyArrays** : Cette option permet de conserver les documents qui n'ont pas de tableau ou qui ont un tableau vide pour le champ concerné.

javascript

Copier le code

{ $unwind: { path: "$items", preserveNullAndEmptyArrays: true } }

1. **includeArrayIndex** : Permet de créer un champ qui enregistre l'indice de chaque élément déplié du tableau.

javascript

Copier le code

{ $unwind: { path: "$items", includeArrayIndex: "arrayIndex" } }

**Exemple avec preserveNullAndEmptyArrays**

Imaginons un document sans éléments dans items :

json

Copier le code

{ "\_id": 3, "customer": "Charlie", "items": [] }

En utilisant $unwind avec preserveNullAndEmptyArrays: true, le document de Charlie sera inclus dans les résultats même s'il n'a pas de produits dans items.

javascript

Copier le code

db.orders.aggregate([

{ $unwind: { path: "$items", preserveNullAndEmptyArrays: true } }

])

**Résultat :**

json

Copier le code

{ "\_id": 3, "customer": "Charlie" } // Document inclus même avec un tableau vide