

Support : MongoDB

Introduction

MongoDB, la base de données NoSQL la plus populaire, est une base de données open source orientée document. Le terme « NoSQL » signifie « non relationnel ». Cela signifie que MongoDB n'est pas basé sur une structure de base de données relationnelle de type table, mais fournit un mécanisme totalement différent pour le stockage et la récupération des données. Ce format de stockage est appelé BSON (similaire au format JSON).

Une structure simple de document MongoDB :

```
{
  titre : 'Support MongoDB',
  par : 'Volatiana Marielle',
  URL : 'https://www.test.org',
  tapez : 'NoSQL'
}
```

Les bases de données SQL stockent les données sous forme de tableau. Ces données sont stockées dans un modèle de données prédéfini qui n'est pas très flexible pour les applications réelles en forte croissance d'aujourd'hui. **Les applications modernes sont plus en réseau, sociales et interactives que jamais**. Les applications stockent de plus en plus de données et y accèdent à des débits plus élevés.

Les systèmes de gestion de bases de données relationnelles (SGBDR) **ne sont pas le bon choix lorsqu'il s'agit de gérer des mégadonnées en raison de leur conception, car ils ne sont pas évolutifs horizontalement**. Si la base de données s'exécute sur un seul serveur, elle atteindra une limite de mise à l'échelle. Les bases de données NoSQL sont plus évolutives et offrent des performances supérieures. MongoDB est une telle base de données NoSQL qui évolue en ajoutant de plus en plus de serveurs et augmente la productivité grâce à son modèle de document flexible.

SGBDR contre MongoDB :

- Le SGBDR a une conception de schéma typique qui montre le nombre de tables et la relation entre ces tables, tandis que MongoDB est orienté document. Il n'y a aucune notion de schéma ou de relation.

- Les transactions complexes ne sont pas prises en charge dans MongoDB car les opérations de jointure complexes ne sont pas disponibles.
- MongoDB permet une structure de document très flexible et évolutive. Par exemple, un document de données d'une collection dans MongoDB peut avoir deux champs tandis que l'autre document de la même collection peut en avoir quatre.
- MongoDB est plus rapide que le SGBDR grâce à des techniques d'indexation et de stockage efficaces.
- Quelques termes sont liés dans les deux bases de données. Ce qu'on appelle Table dans SGBDR s'appelle une Collection dans MongoDB. De même, une ligne est appelée un document et une colonne est appelée un champ. MongoDB fournit un « _id » par défaut (s'il n'est pas fourni explicitement) qui est un nombre hexadécimal de 12 octets qui garantit l'unicité de chaque document. Elle est similaire à la clé primaire du SGBDR.

Caractéristiques de MongoDB :

- **Orienté document** : MongoDB stocke le sujet principal dans un nombre minimal de documents et non en le divisant en plusieurs structures relationnelles comme le SGBDR. Par exemple, il stocke toutes les informations d'un ordinateur dans un seul document appelé Ordinateur et non dans des structures relationnelles distinctes comme le CPU, la RAM, le disque dur, etc.
- **Indexation** : Sans indexation, une base de données devrait scanner chaque document d'une collection pour sélectionner ceux qui correspondent à la requête, ce qui serait inefficace. Ainsi, pour une recherche efficace, l'indexation est indispensable et MongoDB l'utilise pour traiter d'énormes volumes de données en très moins de temps.
- **Évolutivité** : MongoDB évolue horizontalement à l'aide du partitionnement (partitionnement des données sur différents serveurs). Les données sont partitionnées en fragments de données à l'aide de la clé de partition, et ces fragments de données sont répartis uniformément sur les partitions résidant sur de nombreux serveurs physiques. De nouvelles machines

peuvent également être ajoutées à une base de données en cours d'exécution.

- **Réplication et haute disponibilité** : MongoDB augmente la disponibilité des données avec plusieurs copies de données sur différents serveurs. En fournissant une redondance, il protège la base de données des pannes matérielles. Si un serveur tombe en panne, les données peuvent être facilement récupérées à partir d'autres serveurs actifs sur lesquels les données étaient également stockées.
- **Agrégation** : les opérations d'agrégation traitent les enregistrements de données et renvoient les résultats calculés. Elle est similaire à la clause GROUPBY dans SQL. Quelques expressions d'agrégation sont sum, avg, min, max, etc.

Où utilisons-nous MongoDB ?

MongoDB est préféré au SGBDR dans les scénarios suivants :

- **Big Data** : Si vous avez une énorme quantité de données à stocker dans des tables, pensez à MongoDB avant les bases de données RDBMS. MongoDB dispose d'une solution intégrée pour partitionner et partager votre base de données.
- **Schéma instable** : l'ajout d'une nouvelle colonne dans le SGBDR est difficile alors que MongoDB est sans schéma. L'ajout d'un nouveau champ n'affecte pas les anciens documents et sera très simple.
- **Données distribuées** Étant donné que plusieurs copies de données sont stockées sur différents serveurs, la récupération des données est instantanée et sûre même en cas de panne matérielle.

Prise en charge linguistique par MongoDB :

MongoDB fournit actuellement une prise en charge officielle des pilotes pour tous les langages de programmation populaires tels que C, C++, Rust, C#, Java, Node.js, Perl, PHP, Python, Ruby, Scala, Go et Erlang.

Installation de MongoDB :

Allez simplement sur <http://www.mongodb.org/downloads> et sélectionnez votre système d'exploitation parmi [Windows](#) , [Linux](#) , [Mac OS X](#) et Solaris. Une explication détaillée sur l'installation de MongoDB est donnée sur leur site.

Pour Windows, quelques options pour les systèmes d'exploitation 64 bits sont disponibles. Lorsque vous utilisez Windows 7, 8 ou des versions plus récentes, sélectionnez **Windows 64 bits 2008 R2+** . Lorsque vous utilisez Windows XP ou Vista, sélectionnez **Windows 64 bits 2008 R2+ hérité** .

Terminologies

MongoDB peut être appelée comme conteneur pour toutes les collections.

- **La collection** est un ensemble de documents MongoDB. C'est similaire aux tables du SGBDR.
- **Le document** est constitué de champs. Il est similaire à un tuple dans un SGBDR, mais il a ici un schéma dynamique. Les documents de la même collection ne doivent pas nécessairement avoir le même ensemble de champs

Comment Commencer

Après avoir [installé](#) MongoDB, vous pouvez voir tous les fichiers installés dans C:\ProgramFiles\MongoDB\ (emplacement par défaut). Dans le répertoire C:\Program Files\MongoDB\Server\3.2\bin, il y a un tas d'exécutables et une brève description à leur sujet serait :

mongo : L'interface de ligne de commande pour interagir avec la base de données.

mongod : C'est la base de données. Configure le serveur.

mongodump : Il extrait le binaire de la base de données (BSON)

mongoexport : Exporte le document au format Json, CSV

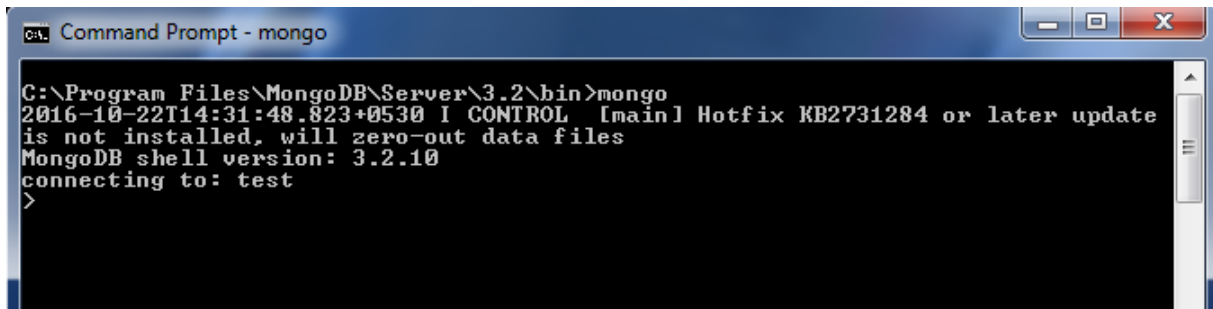
mongoimport : Pour importer certaines données dans la base de données.

mongorestore : pour restaurer tout ce que vous avez exporté.

mongostat : Statistiques des bases de données

Vous pouvez soit copier ce chemin de répertoire dans les variables d'environnement de votre système, soit travailler directement dans le répertoire.

Tapez simplement « mongo » et votre client sera opérationnel et essaiera de se connecter au serveur.



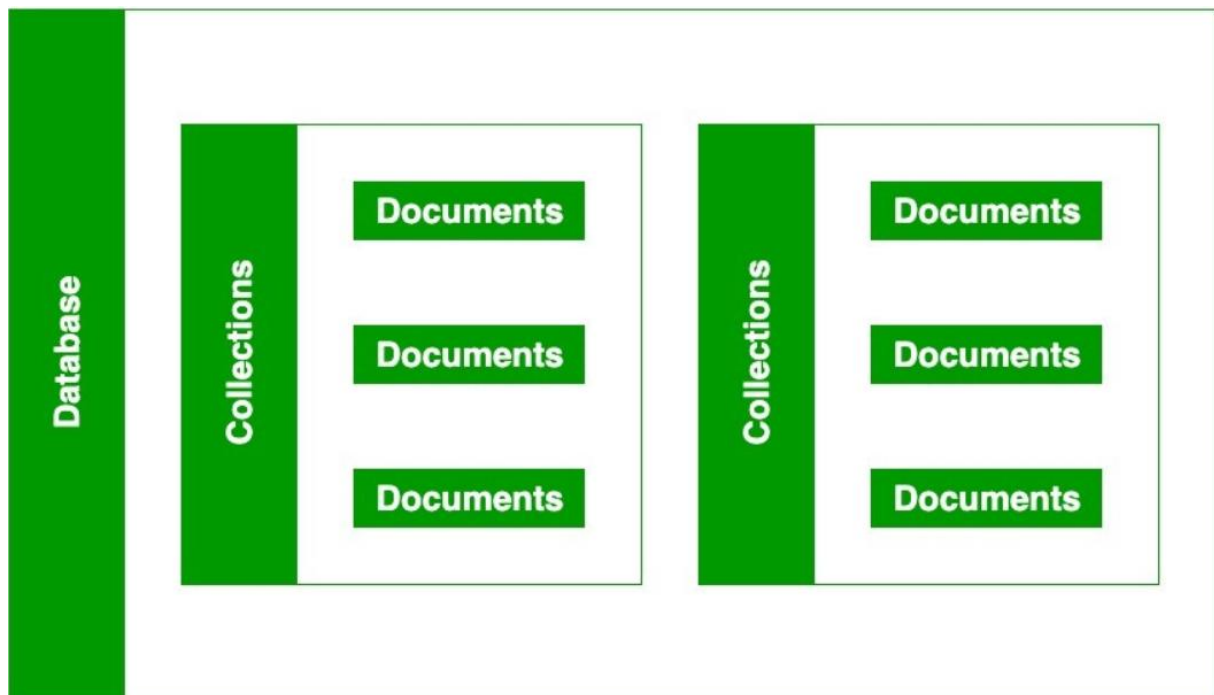
```
C:\Program Files\MongoDB\Server\3.2\bin>mongo
2016-10-22T14:31:48.823+0530 I CONTROL [main] Hotfix KB2731284 or later update
is not installed, will zero-out data files
MongoDB shell version: 3.2.10
connecting to: test
>
```

Ce serait la CLI pour interagir et administrer les bases de données. Ce shell est une sorte de console JS. Vous pouvez essayer différentes commandes JS pour vérifier cela. Puisque notre client est opérationnel, nous pouvons maintenant commencer à travailler sur la base de données. Nous pouvons voir que la base de données utilisée est nommée « test ». Vous pouvez voir les bases de données en utilisant « see dbs » et passer à d'autres bases de données comme « local » en tapant « use <dbname> ».

Notez qu'il n'existe aucune collection existante. Cela peut être vu en tapant la commande « show collections ».

Base de données, collection et document

Les bases de données, les collections et les documents sont des parties importantes de MongoDB. Sans eux, vous ne pourrez pas stocker de données sur le serveur MongoDB. Une base de données contient une collection, et une collection contient des documents et les documents contiennent des données, ils sont liés les uns aux autres.



Base de données

Dans MongoDB, une base de données contient les collections de documents. On peut créer plusieurs bases de données sur le serveur MongoDB.

Afficher la base de données :

Pour voir combien de bases de données sont présentes sur votre serveur MongoDB, écrivez l'instruction suivante dans le shell mongo :

```
show dbs
```

Par exemple:

```
ake product
improvements and to suggest MongoDB products and deployment options to y
ou.

To enable free monitoring, run the following command: db.enableFreeMonit
oring()
To permanently disable this reminder, run the following command: db.disa
bleFreeMonitoring()
---
```

```
> show dbs
admin    0.000GB
config  0.000GB
local   0.000GB
> █
```

Ici, nous avons récemment démarré MongoDB, nous n'avons donc pas de base de données à l'exception de ces trois bases de données par défaut, c'est-à-dire admin, config et local.

Restriction de dénomination pour la base de données :

Avant de créer une base de données, vous devez d'abord vous renseigner sur les restrictions de dénomination des bases de données :

- Dans MongoDB, les noms de base de données ne sont pas sensibles à la casse, mais vous devez toujours vous rappeler que les noms de base de données ne peuvent différer uniquement par la casse des caractères.
- Pour les utilisateurs Windows, les noms de bases de données MongoDB ne peuvent contenir aucun des caractères suivants :

```
/\ . "$* :|?
```

- Pour les utilisateurs Unix et Linux, les noms de bases de données MongoDB ne peuvent contenir aucun des caractères suivants :

```
/\ . "$
```

- Les noms de bases de données MongoDB ne peuvent pas contenir de caractères nuls (dans les systèmes Windows, Unix et Linux).
- Les noms de bases de données MongoDB ne peuvent pas être vides et doivent contenir moins de 64 caractères.

Création d'une base de données :

Dans le shell mongo, vous pouvez créer une base de données à l'aide de la commande suivante :

```
use nom_base
```

Cette commande vous fait basculer vers la nouvelle base de données si le nom donné n'existe pas et si le nom donné existe, elle vous fera basculer vers la base de données existante. Maintenant, à ce stade, si vous utilisez la commande show pour voir la liste des bases de données, vous constaterez que votre nouvelle base de données n'est pas présente dans cette liste de bases de données car, dans MongoDB, la base de données est en fait créée lorsque vous commencez à saisir des données dans cette base de données.

Par exemple:

improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: `db.enableFreeMonitoring()`

To permanently disable this reminder, run the following command: `db.disableFreeMonitoring()`

```
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
[> use GeeksforGeeks
switched to db GeeksforGeeks
[> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
> █
```

Ici, nous créons une nouvelle base de données nommée *GeeksforGeeks* à l'aide de la commande `use`. Après avoir créé une base de données, lorsque nous vérifions la liste des bases de données, nous ne trouvons pas notre base de données sur cette liste car nous ne saisissons aucune donnée dans la base de données *GeeksforGeeks*.

Collection

Les collections sont comme les tables des bases de données relationnelles, elles stockent également des données, mais sous forme de documents. Une seule base de données est autorisée à stocker plusieurs collections.

Sans schéma :

Comme nous le savons, les bases de données MongoDB sont sans schéma. Il n'est donc pas nécessaire dans une collection que le schéma d'un document soit similaire à celui d'un autre document. Ou en d'autres termes, une seule collection contient différents types de documents, comme le montre l'exemple ci-dessous où la collection `mystudentData` contient deux types de documents différents :


```
anki — mongo — 89x17
> db.mystudentData.find().pretty()
[
  {
    "_id" : ObjectId("5e37b67303ab1253cde7afe6"),
    "name" : "Sumit",
    "branch" : "CSE",
    "course" : "DSA",
    "amount" : 4999,
    "paid" : "Yes"
  },
  {
    "_id" : "geeks_for_geeks_201",
    "name" : "Rohit",
    "branch" : "ECE",
    "course" : "Sudo Gate",
    "year" : 2020
  }
]
```

Restrictions de dénomination pour la collection :

Avant de créer une collection, vous devez d'abord vous renseigner sur les restrictions de dénomination des collections :

- Le nom de la collection doit commencer par un trait de soulignement ou un caractère.
- Le nom de la collection ne contient pas de \$, de chaîne vide, de caractère nul et ne commence pas par system. préfixe.
- La longueur maximale du nom de collection est de 120 octets (y compris le nom de la base de données, le séparateur de points et le nom de la collection).

Création d'une collection :

Après avoir créé la base de données, nous créons maintenant une collection pour stocker les documents. La collection est créée en utilisant la syntaxe suivante :

```
db.collection_name.insertOne({..})
```

Ici, la fonction insertOne() est utilisée pour stocker des données uniques dans la collection spécifiée. Et entre accolades {} nous stockons nos données ou en d'autres termes, c'est un document.

Par exemple:

```
anki — mongo — 72x20
[> use GeeksforGeeks
switched to db GeeksforGeeks
[> db.Author.insertOne({name: "Ankita"})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5e3799ff1993ad62dcde4f0d")
}
> █
```

Dans cet exemple, nous créons une collection nommée Author et nous y insérons des données à l'aide de la fonction `insertOne()`. Ou en d'autres termes, `{name: "Ankita"}` est un document de la collection Author, et dans ce document, le nom est la clé ou le champ et "Ankita" est la valeur de cette clé ou de ce champ. Après avoir appuyé sur Entrée, nous avons reçu un message (comme le montre l'image ci-dessus) et ce message nous indique que les données sont entrées avec succès (c'est-à-dire « reconnaître » : vrai) et nous attribue également un identifiant créé automatiquement. C'est la particularité fournie par MongoDB que chaque document fournit un identifiant unique et généralement, cet identifiant est créé automatiquement, mais vous êtes autorisé à créer votre propre identifiant (doit être unique).

Document

Dans MongoDB, les enregistrements de données sont stockés sous forme de documents BSON. Ici, BSON signifie représentation binaire des documents JSON, bien que BSON contienne plus de types de données que JSON. Le document est créé à l'aide de paires champ-valeur ou de paires clé-valeur et la valeur du champ peut être de n'importe quel type BSON.

Syntaxe:

```
{
champ1 : valeur1
champ2 : valeur2
....
champN : valeurN
}
```

Restriction de dénomination des champs :

Avant d'aller plus loin, vous devez vous renseigner sur les restrictions de dénomination des champs :

- Les noms de champs sont des chaînes.
- Le nom du champ `_id` est réservé pour être utilisé comme clé primaire. Et la valeur de ce champ doit être unique, immuable et peut être de tout type autre qu'un tableau.
- Le nom du champ ne peut pas contenir de caractères nuls.
- Les noms de champs de niveau supérieur ne doivent pas commencer par le signe dollar (\$).

Taille du document : La taille maximale du document BSON est de 16 Mo. Cela garantit que le document unique n'utilise pas trop de RAM ou de bande passante (pendant la transmission). Si un document contient plus de données que la taille spécifiée, MongoDB fournit une API GridFS pour stocker ce type de documents.

Notes IMPORTANTES -

- Un seul document peut contenir des champs en double.
- MongoDB enregistre toujours l'ordre des champs dans les documents à l'exception du champ `_id` (qui vient toujours en premier lieu) et le renommage des champs peut modifier l'ordre des champs dans les documents.
- **Champ `_id` :** dans MongoDB, chaque magasin de documents de la collection doit contenir un champ `_id` unique, c'est comme une clé primaire dans une base de données relationnelle. La valeur du champ `_id` peut être définie par l'utilisateur ou par le système (si l'utilisateur ne crée pas de champ `_id`, alors le système générera automatiquement un ObjectId pour le champ `_id`).
 - Lorsque vous créez une collection, MongoDB crée automatiquement un index unique sur le *champ `_id`*.
 - Le champ `_id` est le premier champ de chaque document.
 - La valeur du champ `_id` peut être de n'importe quel type BSON à l'exception des tableaux.
 - La valeur par défaut du champ `_id` est ObjectId.

Exemple 1:

```
anki — mongo — 89x24
. Number of files is 256, should be at least 1000
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible t
o you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring
()
---

> use GeeksforGeeks
switched to db GeeksforGeeks
> db.mystudentData.insertOne({ name: "Sumit", branch: "CSE", course: "DSA", amount: 4999,
paid: "Yes"})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5e37b67303ab1253cde7afe6")
}
```

Ici, le nom, la branche, le cours et le champ payant contiennent des valeurs de type chaîne. Le champ montant contient la valeur de type entier et le champ `_id` est généré par le système.

Exemple n°2 :

```
anki — mongo — 89x24
[> use GeeksforGeeks
switched to db GeeksforGeeks
> db.mystudentData.insertOne({
... _id: "geeks_for_geeks_201",
... name: "Rohit",
... branch: "ECE",
... course: "Sudo Gate",
... year: 2020
... })
{ "acknowledged" : true, "insertedId" : "geeks_for_geeks_201" }
>
```

Ici, le champ `_id` est créé par l'utilisateur.

Opérations MongoDB CRUD

Les opérations CRUD (Créer, Lire, Mettre à jour et Supprimer) constituent l'ensemble d'opérations de base qui permettent aux utilisateurs d'interagir avec le serveur MongoDB.

Comme nous le savons, pour utiliser MongoDB, nous devons interagir avec le serveur MongoDB pour effectuer certaines opérations telles que la saisie de nouvelles données dans l'application, la mise à jour des données dans l'application, la suppression de données de l'application et la lecture des données de l'application.



Maintenant que nous connaissons les composants de l'opération CRUD, découvrons chaque opération individuelle dans MongoDB. Nous saurons ce que fait chaque opération et les méthodes pour effectuer ces opérations dans MongoDB.

Nous créerons, lirons, mettrons à jour et supprimerons des documents du serveur MongoDB.

1. CREATE

Les **opérations de création ou d'insertion** permettent d'insérer ou d'ajouter de nouveaux documents dans la collection. Si une collection n'existe pas, une nouvelle collection sera créée dans la base de données.

Vous pouvez effectuer, créer des opérations à l'aide des méthodes suivantes fournies par MongoDB :

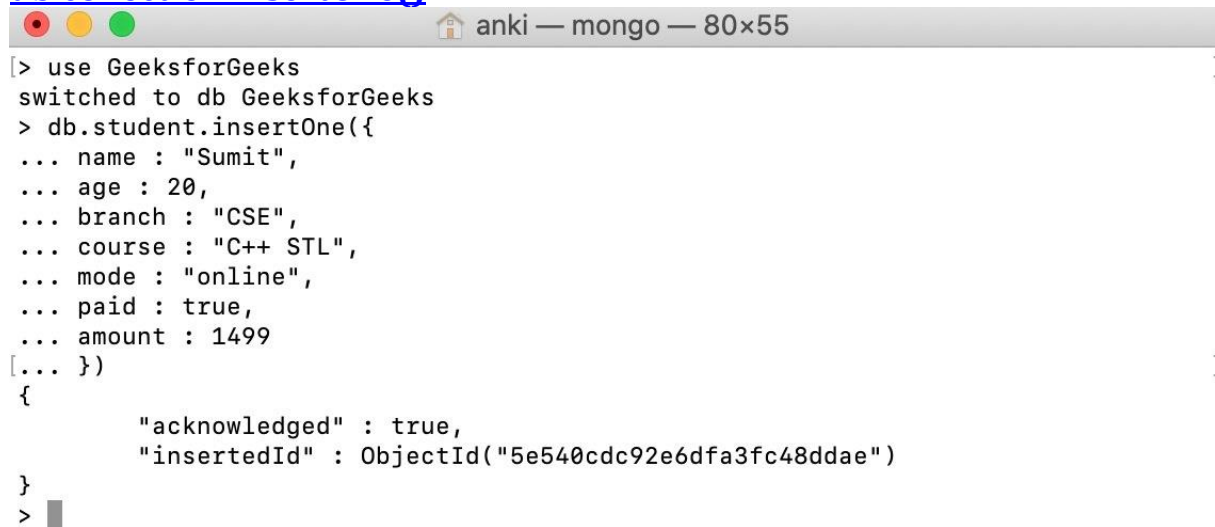
Méthode	Description
db.collection.insertOne()	Il permet d'insérer un seul document dans la collection.
db.collection.insertMany()	Il est utilisé pour insérer plusieurs documents dans la collection.

Méthode	Description
db.createCollection()	Il est utilisé pour créer une collection vide.

Exemple de création d'opérations

Examinons quelques exemples de l'opération Create à partir de CRUD dans MongoDB.

Exemple 1 : Dans cet exemple, nous insérons les détails d'un seul étudiant sous forme de document dans la collection étudiant à l'aide de [la méthode db.collection.insertOne\(\)](#).



```

anki — mongo — 80x55
[> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.insertOne({
... name : "Sumit",
... age : 20,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
[... ]})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5e540cdc92e6dfa3fc48ddae")
}
> █

```

Exemple 2 : Dans cet exemple, nous insérons les détails de plusieurs étudiants sous forme de documents dans la collection étudiant à l'aide de [la méthode db.collection.insertMany\(\)](#).

```
anki — mongo — 80x55
[> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.insertMany([
... {
... name : "Sumit",
... age : 20,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
... },
...
... {
... name : "Rohit",
... age : 21,
... branch : "CSE",
... course : "C++ STL",
... mode : "online",
... paid : true,
... amount : 1499
... }
... ])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5e540d3192e6dfa3fc48ddaf"),
    ObjectId("5e540d3192e6dfa3fc48ddb0")
  ]
}
> █
```

2. READ

Les opérations de lecture sont utilisées pour récupérer des documents de la collection, ou en d'autres termes, les opérations de lecture sont utilisées pour interroger une collection pour un document.

Vous pouvez effectuer une opération de lecture en utilisant la méthode suivante fournie par MongoDB :

Méthode	Description
db.collection.find()	Il permet de récupérer des documents de la collection.

Remarque : la méthode `Pretty()` est utilisée pour décorer le résultat de manière à ce qu'il soit facile à lire.

Exemple d'opérations de lecture

Examinons quelques exemples d'opérations de lecture à partir de CRUD dans MongoDB.

Exemple : Dans cet exemple, nous récupérerons les détails des étudiants de la collection d'étudiants à l'aide de la méthode [db.collection.find\(\)](#).

```
anki — mongo — 80x55
[> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.find().pretty()
]
{
  "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
  "name" : "Sumit",
  "age" : 20,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499
}
{
  "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
  "name" : "Sumit",
  "age" : 20,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499
}
{
  "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
  "name" : "Rohit",
  "age" : 21,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499
}
> █
```

3. UPDATE

Les opérations de mise à jour permettent de mettre à jour ou de modifier le document existant dans la collection. Vous pouvez effectuer des opérations de mise à jour à l'aide des méthodes suivantes fournies par MongoDB :

Méthode	Description
db.collection.updateOne()	Il est utilisé pour mettre à jour un seul document de la collection qui satisfait aux critères donnés.
db.collection.updateMany()	Il est utilisé pour mettre à jour plusieurs documents de la collection qui satisfont aux critères donnés.

Méthode	Description
db.collection.replaceOne()	Il est utilisé pour remplacer un seul document de la collection qui satisfait aux critères donnés.

Exemple d'opérations de mise à jour

Examinons quelques exemples d'opération de mise à jour depuis CRUD dans MongoDB.

Exemple 1 : Dans cet exemple, nous mettons à jour l'âge de Sumit dans la collection `étudiante` à l'aide de la méthode [db.collection.updateOne\(\)](#). **Exemple 2 :** Dans cet exemple, nous mettons à jour l'année de cours dans tous les documents de la collection `étudiant` à l'aide de la méthode [db.collection.updateMany\(\)](#).

```
anki — mongo — 80x43
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.updateOne({name: "Sumit"},{$set:{age: 24 }})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }
> db.student.find().pretty()
{
  "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
  "name" : "Sumit",
  "age" : 24,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499
}
{
  "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
  "name" : "Sumit",
  "age" : 20,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499
}
{
  "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
  "name" : "Rohit",
  "age" : 21,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499
}
> █
```

```
anki — mongo — 80x43
[> use GeeksforGeeks
switched to db GeeksforGeeks
[> db.student.updateMany({}, {$set: {year: 2020}})
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
[> db.student.find().pretty()
{
  "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
  "name" : "Sumit",
  "age" : 24,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499,
  "year" : 2020
}
{
  "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
  "name" : "Sumit",
  "age" : 20,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499,
  "year" : 2020
}
{
  "_id" : ObjectId("5e540d3192e6dfa3fc48ddb0"),
  "name" : "Rohit",
  "age" : 21,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499,
  "year" : 2020
}
> █
```

4. DELETE

L'opération de suppression est utilisée pour supprimer ou supprimer les documents d'une collection. Vous pouvez effectuer des opérations de suppression à l'aide des méthodes suivantes fournies par MongoDB :

Méthode	Description
db.collection.deleteOne()	Il est utilisé pour supprimer un seul document de la collection qui satisfait aux critères donnés.

Méthode	Description
db.collection.deleteMany()	Il est utilisé pour supprimer plusieurs documents de la collection qui satisfont aux critères donnés.

Exemples d'opérations de suppression

Examinons quelques exemples d'opérations de suppression de CRUD dans MongoDB.

Exemple 1 : Dans cet exemple, nous supprimons un document de la collection étudiant à l'aide de la méthode [db.collection.deleteOne\(\)](#).

```

> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.find().pretty()
{
  "_id" : ObjectId("5e540cdc92e6dfa3fc48ddae"),
  "name" : "Sumit",
  "age" : 24,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499,
  "year" : 2020
}
{
  "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
  "name" : "Sumit",
  "age" : 20,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499,
  "year" : 2020
}
{
  "_id" : ObjectId("5e54103592e6dfa3fc48ddb1"),
  "name" : "Rohit",
  "age" : 21,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499
}
> db.student.deleteOne({name: "Sumit"})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.student.find().pretty()
{
  "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
  "name" : "Sumit",
  "age" : 20,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499,
  "year" : 2020
}
{
  "_id" : ObjectId("5e54103592e6dfa3fc48ddb1"),
  "name" : "Rohit",
  "age" : 21,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499
}
[]

```

Exemple 2 : Dans cet exemple, nous supprimons tous les documents de la collection étudiant à l'aide de la méthode [db.collection.deleteMany\(\)](#).

```

anki — mongo — 80x56
> use GeeksforGeeks
switched to db GeeksforGeeks
> db.student.find().pretty()
{
  "_id" : ObjectId("5e540d3192e6dfa3fc48ddaf"),
  "name" : "Sumit",
  "age" : 20,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499,
  "year" : 2020
}
{
  "_id" : ObjectId("5e54103592e6dfa3fc48ddb1"),
  "name" : "Rohit",
  "age" : 21,
  "branch" : "CSE",
  "course" : "C++ STL",
  "mode" : "online",
  "paid" : true,
  "amount" : 1499
}
> db.student.deleteMany({})
{ "acknowledged" : true, "deletedCount" : 2 }
>
```

Agrégation dans MongoDB

Les opérations d'agrégation MongoDB traitent les enregistrements/documents de données et renvoient les résultats calculés. Il collecte les valeurs de divers documents, les regroupe, puis effectue différents types d'opérations sur ces données groupées telles que **sum**, **moyenne**, **minimum**, **maximum**, etc. pour renvoyer un résultat calculé. Elle est similaire à la [fonction d'agrégation de SQL](#).

L'agrégation dans MongoDB permet aux utilisateurs de transformer, filtrer et analyser les données. Ils sont utilisés sur plusieurs documents et constituent un moyen efficace de résumer les données.

MongoDB propose **trois façons** d'effectuer l'agrégation

- [Pipelines d'agrégation](#)
- [Fonction de réduction de carte](#)
- [Agrégation à usage unique](#)

Pipelines d'agrégation

Les pipelines d'agrégation dans MongoDB se composent d'étapes et chaque étape transforme le document. Il s'agit d'un pipeline à plusieurs étapes et dans

chaque état, les documents sont pris en entrée pour produire l'ensemble de documents résultant.

Dans l'étape suivante (ID disponible), les documents résultants sont pris en entrée pour produire la sortie, ce processus se poursuit jusqu'à la dernière étape.

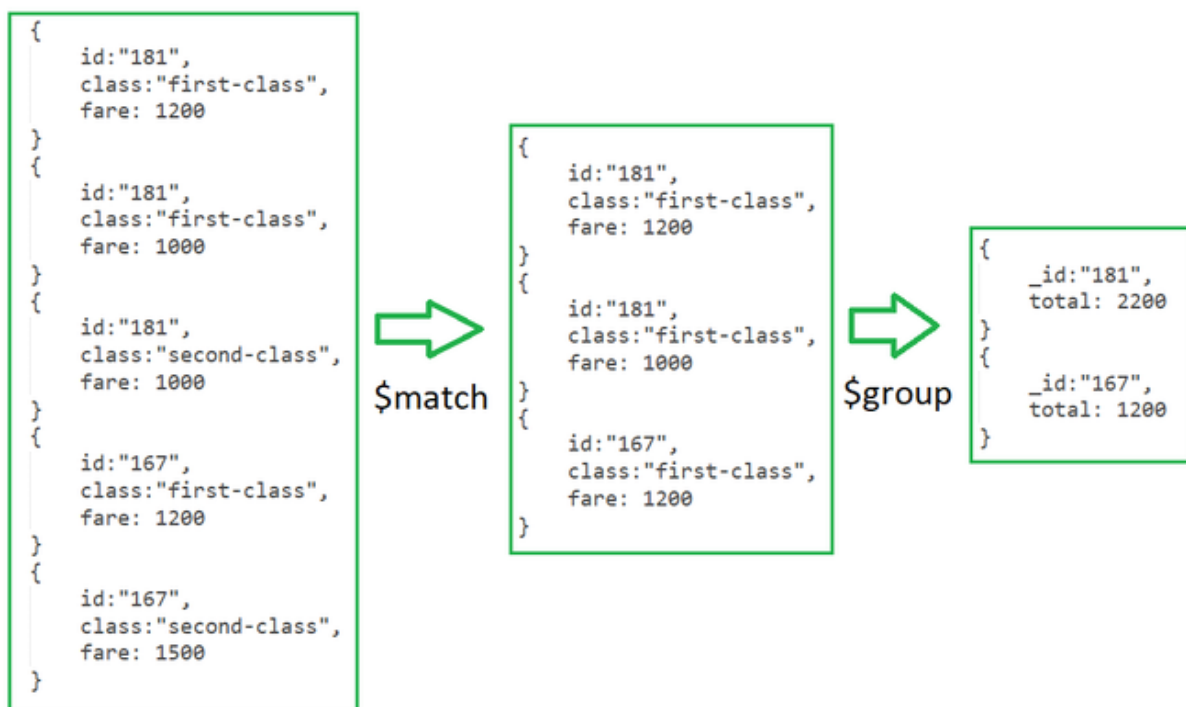
Les **étapes de base du pipeline** fournissent :

- des filtres qui fonctionneront comme des requêtes
- la transformation du document qui modifie le document résultant
- provide pipeline fournit des outils pour **regrouper et trier les documents**.

Le pipeline d'agrégation peut également être utilisé dans [une collection fragmentée](#).

Discutons du pipeline d'agrégation à l'aide d'un exemple :

```
db.train.aggregate( [
  { $match: { class: "first-class" } },
  { $group: { _id: "id", total: { $sum: "$fare" } } } ] pipeline stages
)
```




Explication:

Dans l'exemple ci-dessus d'une collection de « tarifs de train ». **\$match** stage filtre les documents par la valeur du champ de classe, c'est-à-dire classe : « première classe » dans la première étape et transmet le document à la deuxième étape.

Dans la deuxième étape, l' **étape \$group** regroupe les documents par champ id pour calculer la somme du tarif pour chaque identifiant unique.

Ici, la **fonction Aggregate()** est utilisée pour effectuer l'agrégation. Il peut avoir trois opérateurs **étapes**, **expression** et **accumulateur**. Ces opérateurs travaillent ensemble pour atteindre le résultat final souhaité.

```
db.train.aggregate([{$group : { _id : "$id", total : { $sum : "$fare" }}}])
```



Étapes

Chaque étape part des opérateurs d'étape qui sont :

- **\$match** : il est utilisé pour filtrer les documents et réduire la quantité de documents fournis en entrée à l'étape suivante.
- **\$project** : Il est utilisé pour sélectionner certains champs spécifiques d'une collection.
- **\$group** : Il est utilisé pour regrouper des documents en fonction d'une certaine valeur.
- **\$sort** : Il est utilisé pour trier les documents qui les réorganisent
- **\$skip** : Il est utilisé pour sauter n nombre de documents et transmettre les documents restants
- **\$limit** : Il est utilisé pour transmettre le premier nombre n de documents, les limitant ainsi.
- **\$unwind** : il est utilisé pour dérouler les documents qui utilisent des tableaux, c'est-à-dire qu'il déconstruit un champ de tableau dans les documents pour renvoyer des documents pour chaque élément.
- **\$out** : Il est utilisé pour écrire les documents résultants dans une nouvelle collection

Expressions

Il fait référence au nom du champ dans les documents d'entrée pour par exemple { \$group : { _id : " \$id ", total:{ \$sum:" \$fare "}}} ici \$id et \$fare sont des expressions.

Accumulateurs

Ceux-ci sont essentiellement utilisés en phase de groupes

- **sum** : il additionne les valeurs numériques des documents de chaque groupe
- **compte** le compte le nombre total de documents
- **avg** : il calcule la moyenne de toutes les valeurs données à partir de tous les documents
- **min** : il obtient la valeur minimale de tous les documents
- **max** : il obtient la valeur maximale de tous les documents
- **first** : il récupère le premier document du regroupement
- **last** : il récupère le dernier document du regroupement

Note:

- dans \$group, _id est un champ obligatoire
- \$out doit être la dernière étape du pipeline
- \$sum:1 comptera le nombre de documents et \$sum:"\$fare" donnera la somme du tarif total généré par identifiant.

Exemples de pipeline d'agrégation MongoDB

Dans les exemples suivants, nous travaillons avec :

Base de données : *GeeksForGeeks*

Collection : *étudiants*

Documents : *Sept documents contenant les détails des étudiants sous forme de paires champ-valeur.*

C:\WINDOWS\system32\cmd.exe - mongo

```
> db.students.find().pretty()
```

```
{
  "_id" : ObjectId("6024aefbf54bd0745f0db733"),
  "name" : "tony",
  "age" : 17,
  "id" : 1,
  "sec" : "A",
  "subject" : [
    "physics",
    "maths"
  ]
}
{
  "_id" : ObjectId("6024aefbf54bd0745f0db734"),
  "name" : "steve",
  "age" : 37,
  "id" : 2,
  "sec" : "A"
}
{
  "_id" : ObjectId("6024aefbf54bd0745f0db735"),
  "name" : "natasha",
  "age" : 17,
  "id" : 3,
  "sec" : "B",
  "subject" : [
    "physics",
    "english"
  ]
}
{
  "_id" : ObjectId("6024aefbf54bd0745f0db736"),
  "name" : "bruce",
  "age" : 21,
  "id" : 4,
  "sec" : "B",
  "subject" : [
    "physics",
    "maths",
    "biology",
    "Chemistry"
  ]
}
{
  "_id" : ObjectId("6024aefbf54bd0745f0db737"),
  "name" : "nick",
  "age" : 40,
  "id" : 5,
  "sec" : "B",
  "subject" : [
    "english"
  ]
}
{
  "_id" : ObjectId("6024aefbf54bd0745f0db738"),
  "name" : "groot",
  "age" : 4,
  "id" : 6,
  "sec" : "A",
  "subject" : [
    "english"
  ]
}
{
  "_id" : ObjectId("6024aefbf54bd0745f0db739"),
  "name" : "thanos",
  "age" : 4,
  "id" : 7,
  "sec" : "A",
  "subject" : [
    "maths",
    "physics",
    "chemistry"
  ]
}
```

Exemple 1

Affichage du nombre total d'élèves dans la section B.

```
db.students.aggregate([{$match:{sec:"B"}},{ $count:"Total étudiant en sec:B"}])
```

Sortir:

```
> db.students.aggregate([{$match:{sec:"B"}},{ $count:"Total student in sec:B"}])
{ "Total student in sec:B" : 3 }
> -
```

Explication:

Dans cet exemple, pour compter le nombre d'étudiants dans la section B, nous filtrons d'abord les documents à l'aide de l' **opérateur \$match**, puis nous utilisons l' accumulateur **\$count** pour compter le nombre total de documents transmis après filtrage à partir du \$ correspondre.

Exemple 2

Affichage du nombre total d'élèves dans les deux sections et de l'âge maximum des deux sections

```
db.students.aggregate([{$group : {_id:"$sec", total_st : {$sum:1}, max_age :{$max:"$age"} } }])
```

Sortir

```
> db.students.aggregate([{$group: {_id:"$sec", total_st: {$sum:1}, max_age:{$max:"$age"} } }])
{ "_id" : "A", "total_st" : 4, "max_age" : 37 }
{ "_id" : "B", "total_st" : 3, "max_age" : 40 }
>
```

Explication

Dans cet exemple, nous utilisons **\$group** pour grouper, afin de pouvoir compter pour toutes les autres sections des documents. Ici, **\$sum** résume le document dans chaque groupe et l'accumulateur **\$max** est appliqué à l'expression d'âge qui trouvera l'âge maximum dans chaque document.

Carte Réduire

[La réduction de carte](#) est utilisée pour agréger les résultats pour un grand volume de données.

Map réduire a deux fonctions principales : **mapper** pour regrouper tous les documents et **réduire** pour effectuer une opération sur les données groupées.

Syntaxe

```
db.collectionName.mapReduce(mappingFunction, reductionFunction,  
{out: 'Result'});
```

Exemple de réduction de carte MongoDB

Dans l'exemple suivant, nous travaillons avec :

Base de données : GeeksForGeeks

Collection : StudentsMark

Documents : Sept documents contenant les détails des étudiants sous la forme de paires champ-valeur.

```
C:\WINDOWS\system32\cmd.exe - mongo
> db.studentsMark.find().pretty()
{
  "_id" : ObjectId("60256038d423257579040c29"),
  "name" : "tony",
  "age" : 17,
  "marks" : 30
}
{
  "_id" : ObjectId("60256038d423257579040c2a"),
  "name" : "bruce",
  "age" : 17,
  "marks" : 40
}
{
  "_id" : ObjectId("60256038d423257579040c2b"),
  "name" : "steve",
  "age" : 27,
  "marks" : 39
}
{
  "_id" : ObjectId("60256038d423257579040c2c"),
  "name" : "bucky",
  "age" : 27,
  "marks" : 16
}
{
  "_id" : ObjectId("60256038d423257579040c2d"),
  "name" : "nick",
  "age" : 37,
  "marks" : 30
}
{
  "_id" : ObjectId("60256038d423257579040c2e"),
  "name" : "loki",
  "age" : 19,
  "marks" : 30
}
{
  "_id" : ObjectId("60256038d423257579040c2f"),
  "name" : "groot",
  "age" : 37,
  "marks" : 20
}
```

Exemple

Calculer la somme des notes pour chaque tranche d'âge

```
var mapfunction = function(){emit(this.age, this.marks)}
var réduirefunction = function(clé, valeurs){return
Array.sum(values)}
db.studentsMarks.mapReduce(mapfunction, réductionfunction, {'out'
:'Résultat'})
```

```
C:\WINDOWS\system32\cmd.exe - mongo
```

```
> var mapfunction = function(){emit(this.age,this.marks)}  
> var reducefunction = function(key,values){return Array.sum(values)}  
> db.studentsMark.mapReduce(mapfunction,reducefunction,{ 'out': 'Results' })  
{ "result" : "Results", "ok" : 1 }  
> db.Results.find()  
{ "_id" : 19, "value" : 30 }  
{ "_id" : 27, "value" : 55 }  
{ "_id" : 37, "value" : 50 }  
{ "_id" : 17, "value" : 70 }  
> _
```

Explication

La première fonction (`mapfunction`) attribue l'âge (comme « `_id` » plus tard) et marque comme une paire clé-valeur pour chaque élève. Ces données sont ensuite transmises à la deuxième fonction (`reducefunction`), qui regroupe les élèves par âge et calcule probablement la somme de leurs notes à l'aide d'une fonction distincte. Les résultats finaux sont stockés dans une nouvelle collection nommée « Résultats ».

Agrégation à usage unique

Il est utilisé lorsque nous avons besoin d'un accès simple au document, comme compter le nombre de documents ou pour trouver toutes les valeurs distinctes dans un document.

Il fournit simplement l'accès au processus d'agrégation commun à l'aide des méthodes **`count()`** , **`distinct()`** et **`estimateDocumentCount()`** , ce qui lui manque donc la flexibilité et les capacités du pipeline.

Exemple d'agrégation MongoDB à usage unique

Dans l'exemple suivant, nous travaillons avec :

Base de données : *GeeksForGeeks*

Collection : *StudentsMark*

Documents : *Sept documents contenant les détails des étudiants sous la forme de paires champ-valeur.*

```
C:\WINDOWS\system32\cmd.exe - mongo
> db.studentsMark.find().pretty()
{
  "_id" : ObjectId("60256038d423257579040c29"),
  "name" : "tony",
  "age" : 17,
  "marks" : 30
}
{
  "_id" : ObjectId("60256038d423257579040c2a"),
  "name" : "bruce",
  "age" : 17,
  "marks" : 40
}
{
  "_id" : ObjectId("60256038d423257579040c2b"),
  "name" : "steve",
  "age" : 27,
  "marks" : 39
}
{
  "_id" : ObjectId("60256038d423257579040c2c"),
  "name" : "bucky",
  "age" : 27,
  "marks" : 16
}
{
  "_id" : ObjectId("60256038d423257579040c2d"),
  "name" : "nick",
  "age" : 37,
  "marks" : 30
}
{
  "_id" : ObjectId("60256038d423257579040c2e"),
  "name" : "loki",
  "age" : 19,
  "marks" : 30
}
{
  "_id" : ObjectId("60256038d423257579040c2f"),
  "name" : "groot",
  "age" : 37,
  "marks" : 20
}
```

Exemple 1

Affichage de noms et d'âges distincts (non répétitifs)

```
db.studentsMarks.distinct("nom")
```

Sortir:

```
C:\WINDOWS\system32\cmd.exe - mongo
> db.studentsMark.distinct("name")
[ "bruce", "bucky", "groot", "loki", "nick", "steve", "tony" ]
> db.studentsMark.distinct("age")
[ 17, 19, 27, 37 ]
> _
```

Explication:

Ici, nous utilisons une **méthode distinct()** qui trouve des valeurs distinctes du champ spécifié (c'est-à-dire le nom).

Exemple 2

Compter le nombre total de documents

```
db.studentsMarks.count()
```

Sortir

```
C:\WINDOWS\system32\cmd.exe - mongo
> db.studentsMark.count()
7
> _
```

Explication:

Ici, nous utilisons count() pour trouver le numéro total du document, contrairement à la méthode find(), elle ne trouve pas tous les documents mais les compte et renvoie un nombre.