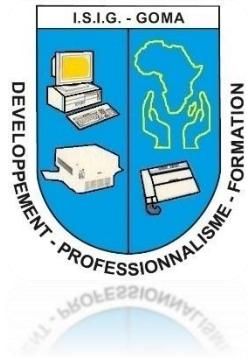


**REPUBLIQUE DEMOCRATIQUE DU CONGO
ENSEIGNEMENT SUPERIEUR ET UNIVERSITAIRE**

INSTITUT SUPERIEUR D'INFORMATIQUE ET DE GESTION

ISIG



B.P. 841 GOMA

**Base de données non relationnelle,
Data warehousing et Big Data**

Dispensé Par : Ir. Maley Musema gloDi

Promotion : LIC3 P LIAGE

V.H : 90H = 6 Crédits

Année Académique : 2022 - 2023

CONTEXTE

Dans ce cours intitulé **Base de données non relationnelle, Data warehousing et Big Data** ; nous allons aborder deux concepts dont le concept **Base de données non relationnelle** et le concept **Data warehousing et Big Data**.

Les deux concepts vont constituer deux grandes parties du cours, et chaque partie avec quelques chapitres pour illustrer beaucoup plus de notions par rapport aux concepts.

OBJECTIF DU COURS

L'étudiant qui l'aura suivi avec attention, aura des compétences ci-après :

- Comprendre les notions de base aux Base de données non SQL
- Appréhender à manipuler les SGBD non Sql, notamment MongoDB
- Être capable de créer, de Manipuler, et de gérer une base de données non Sql.
- Comprendre les notions de base en programmation Arduino.
- Comprendre les notions de base sur le concept entrepôt de données (Data warehouse).

PARTIE 1 : BASE DE DONNÉES NON RELATIONNELLE

INTRODUCTION

Le terme « NoSQL » désigne les différents types de bases de données non relationnelles. Ces bases de données stockent les données dans un format différent. Toutefois, les bases de données NoSQL peuvent être interrogées à l'aide d'API en langage idiomatique, de langages déclaratifs et de langages de requête par exemple, ce qui explique pourquoi elles sont également considérées comme des bases de données « pas seulement SQL ».

À quoi sert une base de données NoSQL ?

Les bases de données NoSQL sont largement utilisées dans les applications Web et le big data en temps réel, car elles présentent le principal avantage de proposer une évolutivité élevée et une haute disponibilité.

Les bases de données NoSQL sont généralement préférées par les développeurs, car elles se prêtent naturellement à un paradigme de développement agile en s'adaptant rapidement à l'évolution des exigences. **Les bases de données NoSQL permettent de stocker les données de manière plus intuitive et plus facile à comprendre, ou plus proche de la façon dont elles sont utilisées par les applications, avec moins de transformations requises lors du stockage ou de l'extraction à l'aide d'API de type NoSQL.** De plus, les bases de données NoSQL peuvent tirer pleinement parti du cloud pour éviter tout temps d'inactivité.

Quelle est la différence entre le SQL et le NoSQL ?

Les bases de données SQL sont relationnelles, tandis que les bases de données NoSQL ne le sont pas. Le système de gestion de bases de données relationnelle (SGBDR) est la base du langage SQL (Structured Query Language), qui permet aux utilisateurs d'accéder à des données dans des tables hautement structurées et de les manipuler. Il s'agit d'un modèle fondamental pour les systèmes de base de données tels que MS SQL Server, IBM DB2, Oracle et MySQL. Toutefois, avec les bases de données NoSQL, la syntaxe d'accès aux données peut être différente d'une base de données à l'autre.

Quelle est la différence entre les bases de données relationnelles et les bases de données NoSQL ?

Pour comprendre les bases de données NoSQL, il est important de connaître la différence entre le SGBDR et les types de bases de données non relationnelles.

Les données d'un SGBDR sont stockées dans des objets de base de données appelés tables. Une table est un ensemble d'entrées de données associées, qui se compose de colonnes et de lignes. **Ces bases de données nécessitent de définir le schéma à l'avance**, c'est-à-dire que toutes les colonnes et leurs types de données associés doivent être **connus au préalable** afin que

les applications puissent écrire des données dans la base. Elles stockent également des **informations qui relient plusieurs tables par le biais de clés**, créant ainsi une relation entre plusieurs tables. Dans le cas le plus simple, une clé est utilisée pour extraire une ligne spécifique afin de pouvoir l'examiner ou la modifier.

Au contraire, dans les bases de données NoSQL, **les données peuvent être stockées sans définir le schéma à l'avance**, ce qui signifie que vous avez la possibilité d'avancer rapidement, en **définissant le modèle de données au fur et à mesure**. Cette approche peut être adapté à des besoins de votre entreprise, qu'il s'agisse de traiter un répertoire de données de graphes, orientées colonne, orientées document ou clé-valeur.

Jusqu'à récemment, les bases de données relationnelles étaient les modèles les plus utilisés. Elles sont encore particulièrement omniprésentes dans de nombreuses entreprises. Toutefois, la diversité, **la rapidité et le volume des données accessible** aujourd'hui nécessitent parfois une base de données très différente pour compléter la base de données relationnelle. Cette évolution a déclenché l'adoption dans certains domaines des bases de données NoSQL, également appelés « bases de données non relationnelles ». En raison de leur capacité à évoluer horizontalement et rapidement, les bases de données non relationnelles peuvent gérer un trafic élevé, ce qui les rend également très adaptables.

Quand choisir une base de données NoSQL ?

Alors que les entreprises et les organisations ont besoin d'innover rapidement, il est essentiel pour elles de pouvoir rester agiles et continuer à travailler à n'importe quelle échelle. Les bases de données NoSQL proposent des schémas flexibles et prennent également en charge une variété de modèles de données idéaux pour créer des applications qui nécessitent de grands volumes de données et des **temps de latence ou de réponse faibles** (par exemple, des jeux en ligne et des applications Web d'e-commerce).

Quand ne pas choisir une base de données NoSQL ?

Les bases de données NoSQL reposent généralement sur des **données non normalisées**, prenant en charge les types **d'application qui utilisent moins de tables** (ou de conteneurs) et dont les relations de données ne sont pas modélisées à l'aide de références, mais plutôt en tant qu'enregistrements (ou documents). **De nombreuses applications métier back-office classiques de la finance, de la comptabilité et de la planification des ressources d'entreprise reposent sur des données hautement normalisées pour prévenir les anomalies de données et les doublons de données**. Il s'agit généralement des types d'application qui ne conviennent pas bien à une base de données NoSQL.

La complexité des requêtes est également un facteur à prendre en compte pour les bases de données NoSQL. **Elles se révèlent très efficaces pour les requêtes portant sur une seule table. Toutefois, quand les requêtes se complexifient (Bulletin, livre de caisse, Fiche de Stock), les bases de données relationnelles constituent un meilleur choix.** Les bases de données NoSQL n'offrent généralement pas de **jointures complexes, de sous-requêtes et d'imbrication de requêtes** dans une clause WHERE.

Cependant, il n'est parfois pas nécessaire de choisir entre les bases de données relationnelles et non relationnelles. De nombreuses entreprises ont opté pour des bases de données qui proposent un modèle convergé dans lequel elles peuvent combiner des modèles de données relationnels et non relationnels. Cette approche hybride offre une plus grande flexibilité dans la gestion de différents types de données, tout en assurant la cohérence en lecture et en écriture sans dégrader les performances.

Qu'est-ce que les bases de données NoSQL offrent de plus que les autres ?

L'un des principaux facteurs de différenciation entre les bases de données NoSQL et d'autres types de base de données est que les bases de données NoSQL utilisent généralement **un stockage non structuré**. Développées au cours des vingt dernières années, les bases de données NoSQL ont été conçues pour des **requêtes rapides et simples**, des données **volumineuses et des modifications fréquentes des applications**. En outre, ces bases de données **simplifient également le travail des développeurs**.

Un autre facteur de différenciation important est que les bases de données NoSQL s'appuient sur un processus appelé « sharding » **pour évoluer horizontalement**, ce qui signifie qu'un plus grand nombre de machines peuvent être ajoutées pour gérer les données sur plusieurs serveurs. La mise à l'échelle verticale proposée par d'autres bases de données SQL nécessite l'ajout de puissance et de mémoire à la machine existante, ce qui peut ne pas être durable car de plus en plus de stockage est nécessaire.

La nature horizontale du redimensionnement avec les bases de données NoSQL signifie qu'elles peuvent gérer des quantités extrêmement importantes de données, même en cas d'augmentation de leur volume, de manière plus efficace. Il peut être utile de penser à la mise à l'échelle verticale comme l'ajout d'un nouvel étage à votre maison, alors que la mise à l'échelle horizontale est comme la construction d'une autre maison juste à côté de la première.

Les avantages d'une base de données NoSQL

La rapidité et l'ampleur sans précédent de l'interaction numérique et de la consommation de données constatées au cours des vingt dernières années ont nécessité aux entreprises d'adopter une approche plus moderne et fluide du stockage et de la consommation des données. Alors que les utilisateurs du monde entier exigent un flux ininterrompu de contenus et de fonctions, il n'est

pas étonnant que les bases de données aient dû s'adapter rapidement. Dans cette optique, voici quelques-unes des principales raisons pour lesquelles les développeurs choisissent des bases de données NoSQL/non relationnelles :

- **Flexibilité**

Avec les bases de données SQL, les données sont stockées dans une structure bien plus rigide et prédefinie. Toutefois, dans les bases de données NoSQL, les données peuvent être stockées de façon plus libre sans ces schémas rigides. Cette conception permet d'innover et de développer rapidement des applications. Les développeurs peuvent se concentrer sur la création de systèmes pour mieux servir leurs clients sans se soucier des schémas. Les bases de données NoSQL peuvent facilement gérer tous les formats de données, tels que des données structurées, semi-structurées et non structurées, dans un seul et même répertoire.

- **Évolutivité**

Au lieu d'évoluer en ajoutant d'autres serveurs, les bases de données NoSQL peuvent évoluer en utilisant du matériel de base. Il est donc possible de prendre en charge une augmentation du trafic afin de répondre à la demande sans aucun temps d'arrêt. En évoluant, les bases de données NoSQL peuvent devenir plus volumineuses et plus puissantes, c'est pourquoi elles sont devenues l'option privilégiée pour l'évolution des ensembles de données.

- **De hautes performances**

L'architecture évolutive d'une base de données NoSQL peut être particulièrement utile lorsque le volume de données ou le trafic augmente. Comme le montre le graphique ci-dessous, cette architecture garantit des temps de réponse rapides et prévisibles en quelques millisecondes. Les bases de données NoSQL peuvent également ingérer des données et les livrer rapidement et de manière fiable, c'est pourquoi les bases de données NoSQL sont utilisées dans des applications qui collectent des téraoctets de données chaque jour, tout en nécessitant une expérience utilisateur hautement interactive. Dans le graphique ci-dessous, nous affichons un taux entrant de 300 lectures par seconde (ligne bleue) avec une latence du 95e percentile dans la plage des 3 à 4 ms et un taux entrant de 150 écritures par seconde (ligne verte) pour une latence du 95e percentile dans la plage des 4 à 5 ms.

- **Disponibilité**

Les bases de données NoSQL répliquent automatiquement les données sur plusieurs serveurs, data centers ou ressources cloud. Dès lors, la latence pour les utilisateurs est réduite, quelle que soit leur localisation. Cette fonctionnalité permet également de réduire le fardeau que représente la gestion de base de données et permet aux équipes informatiques de se concentrer sur d'autres tâches plus importantes.

- **Hautement fonctionnel**

Les bases de données NoSQL sont conçues pour créer des répertoires de données distribués permettant le stockage de données extrêmement volumineuses. Le NoSQL constitue donc le choix idéal pour les big data, les applications Web en temps réel, les relations client, l'e-commerce, les jeux en ligne, l'Internet des objets, les réseaux sociaux et les applications de publicité en ligne.

Les types de bases de données NoSQL

Il existe quatre principaux types de base de données NoSQL :

- **Valeur clé**

Il s'agit du type de base de données NoSQL le plus flexible, car l'application dispose d'un contrôle total sur ce qui est stocké dans le champ de valeur sans aucune restriction.

Ex : Redis, Memcached.

- **Document**

Également appelées bases de données orientées documents ou répertoires de documents, ces bases de données sont utilisées pour stocker, extraire et gérer des données semi-structurées. Il n'est pas nécessaire de spécifier les champs qu'un document contiendra.

Ex : MongoDB, Elasticsearch

- **Graphe**

Ce type de bases de données organise les données en tant que nœuds et relations, qui indiquent les connexions entre ces nœuds. Il prend en charge une représentation plus riche et plus complète des données. Les bases de données de graphes sont utilisées pour les réseaux sociaux, les systèmes de réservation et la détection des fraudes.

Ex : Neo4j, InfluxDB

- **Colonne large**

Ces bases de données stockent et gèrent les données sous forme de tables, de lignes et de colonnes. Elles sont largement déployées dans des applications qui nécessitent un format de colonne pour capturer des données sans schéma.

Ex : Cassandra, AWS DynamoDB, HBase

Pour le cadre de notre cours, nous allons utiliser **MongoDB** comme SGBD pour aborder les notions sur les bases de données non relationnelles.

MongoDB est une base de données NoSQL open source orientée document. C'est l'une des bases de données NoSQL les plus populaires et les plus utilisées.

MongoDB est une base de données documentaire open source, multiplateforme et distribuée conçue pour faciliter le développement et la mise à l'échelle d'applications. Il s'agit d'une base de données NoSQL développée par MongoDB.

Le nom MongoDB est dérivé du mot "Humongous" qui signifie énorme, énorme. La base de données MongoDB est conçue pour stocker une énorme quantité de données et fonctionner rapidement.

MongoDB n'est pas un système de gestion de base de données relationnelle (RDBMS). C'est ce qu'on appelle une base de données "NoSQL". C'est le contraire des bases de données basées sur SQL où il ne normalise pas les données sous des schémas et des tables où chaque table a une structure fixe. Au lieu de cela, il stocke les données dans les collections sous forme de documents basés sur JSON et n'applique pas de schémas. Il n'a pas de tables, de lignes et de colonnes comme les autres bases de données SQL (RDBMS).

Le tableau suivant répertorie la relation entre les terminologies MongoDB et RDBMS.

MongoDB (base de données NoSQL)	SGBDR (SQL Server, Oracle, etc.)
Base de données	Base de données
Collection	Tableau
Document	Ligne (enregistrement)
Champ	Colonne

Dans la base de données RDBMS, une table peut avoir plusieurs lignes et colonnes. De même dans MongoDB, une collection peut avoir plusieurs documents équivalents aux lignes. Chaque document a plusieurs "champs" qui sont équivalents aux colonnes. Les documents d'une même collection peuvent avoir différents champs.

Voici un exemple de document basé sur JSON.

```
{
  _id: ObjectId("32521df3f4948bd2f54218"),
  firstName: "John",
  lastName: "King",
  email: "john.king@abc.com",
  salary: "33000"
}
```

Le système de gestion de base de données MongoDB a quelques **avantages** suivants :

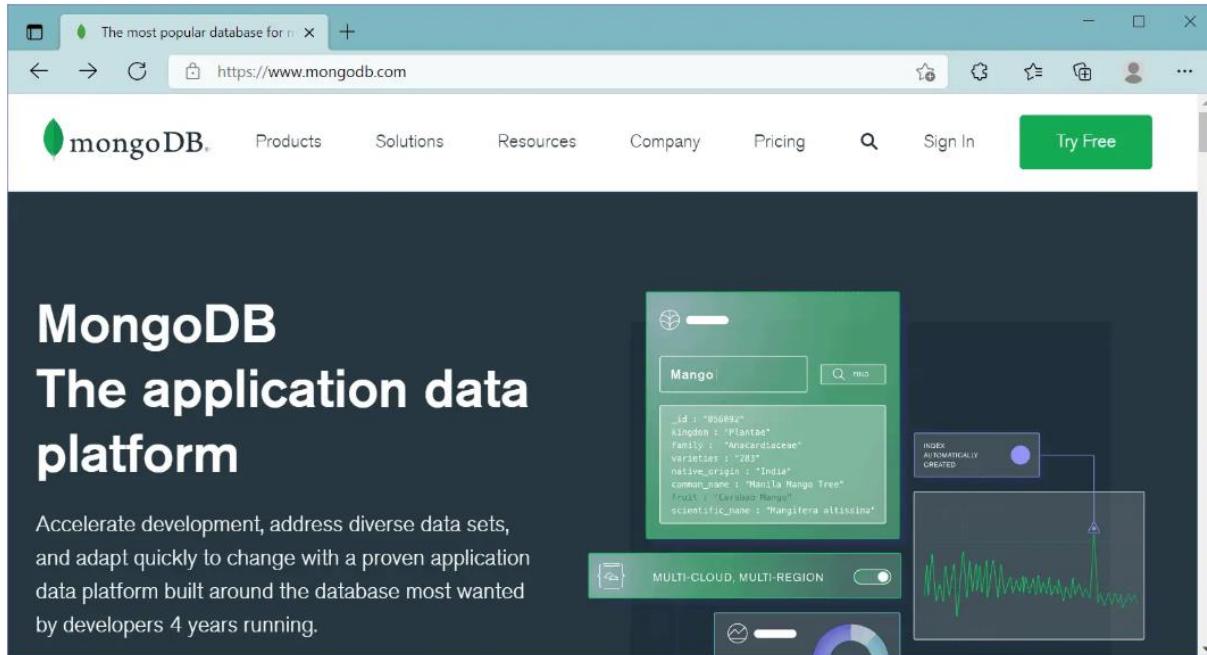
1. MongoDB stocke les données sous forme de document basé sur JSON qui n'applique pas le schéma. Il nous permet de stocker des données hiérarchiques dans un document. Cela facilite le stockage et la récupération des données de manière efficace.
2. Il est facile d'augmenter ou de réduire l'échelle selon les besoins car il s'agit d'une base de données basée sur des documents. MongoDB nous permet également de répartir les données sur plusieurs serveurs.
3. MongoDB fournit des fonctionnalités riches telles que l'indexation, l'agrégation, le magasin de fichiers, etc.
4. MongoDB fonctionne rapidement avec d'énormes données.
5. MongoDB fournit des pilotes pour stocker et récupérer des données à partir de différentes applications développées dans différentes technologies telles que C#, Java, Python, Node.js, etc.
6. MongoDB fournit des outils pour gérer les bases de données MongoDB.

CHAPITRE 1 : INSTALLATION DES OUTILS

Installer MongoDB Server, MongoDB Shell, Compass sous Windows

Ici, vous apprendrez à installer le serveur MongoDB, MongoDB Shell et la boussole sur votre machine Windows locale.

Visitez www.mongodb.com pour télécharger le programme d'installation de MongoDB pour votre plate-forme requise.



Site officiel de MongoDB

Ici, nous allons installer un serveur de base de données MongoDB gratuit sur notre machine Windows locale. Alors, cliquez sur le menu Produit -> Serveur communautaire, comme indiqué ci-dessous.

The screenshot shows the MongoDB homepage. At the top, there's a navigation bar with links for Products, Solutions, Resources, Company, Pricing, a search bar, Sign In, and a Try Free button. Below the header, there's a main menu with sections for Atlas, Enterprise, Community, and Realm. Under the Enterprise section, 'Advanced' is selected. Under Community, 'Community Server' is highlighted with a red box. To the right, there's a sidebar for Tools, including Compass, Database, Shell, VS Code, Plugin, Atlas CLI, and Database Connectors. A large banner on the left side of the page features the text 'MongoDB The app platform' and 'Accelerate development to change with a platform database most wanted'. A 'Start free' button is visible at the bottom of the banner.

Édition communautaire de MongoDB

Cela ouvrira une page de téléchargement où vous pourrez sélectionner les options de version, de plate-forme et de package.

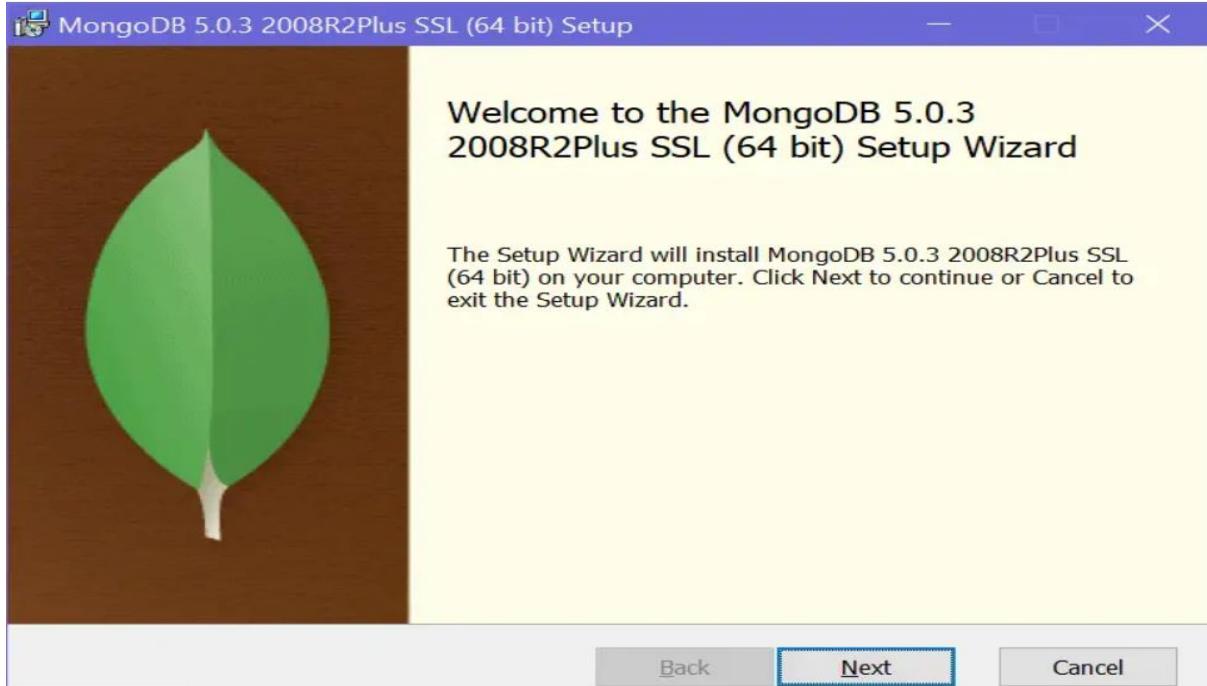
Ici, nous allons télécharger la dernière version de MongoDB, la plate-forme Windows et le fichier msi sous forme de package, comme indiqué ci-dessous.

The screenshot shows the MongoDB Community Download page. At the top, there's a navigation bar with links for Products, Solutions, Resources, Company, Pricing, a search bar, Sign In, and a Try Free button. Below the header, there are four main categories: Atlas (MongoDB as a service), On-premises (MongoDB locally), Tools (Boost productivity), and Mobile & Edge (Realm Datastore). The 'On-premises' section is currently active. On the right side, there's a 'MongoDB Community Server' section with a brief description and a 'Give it a try with a free, highly-available 512 MB cluster.' link. Below this, there's a 'Available Downloads' section with dropdown menus for Version (5.0.3 (current)), Platform (Windows), and Package (msi). A large 'Download' button is highlighted with a red box. At the bottom, there are links for Current releases & packages, Development releases, and Archived releases.

Télécharger MongoDB

Cliquez sur le bouton Télécharger pour télécharger le fichier d'installation.

Une fois entièrement téléchargé, cliquez sur le fichier msi pour démarrer l'assistant d'installation, comme indiqué ci-dessous.



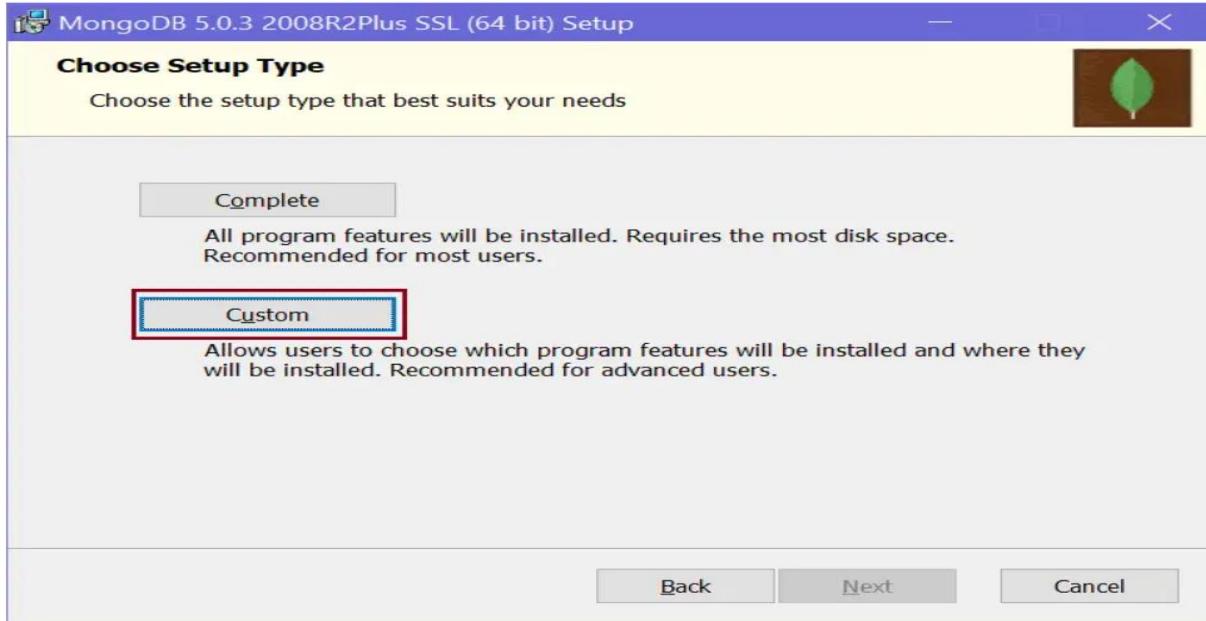
Assistant d'installation de MongoDB

Cliquez sur Suivant pour démarrer l'installation.



Assistant d'installation de MongoDB

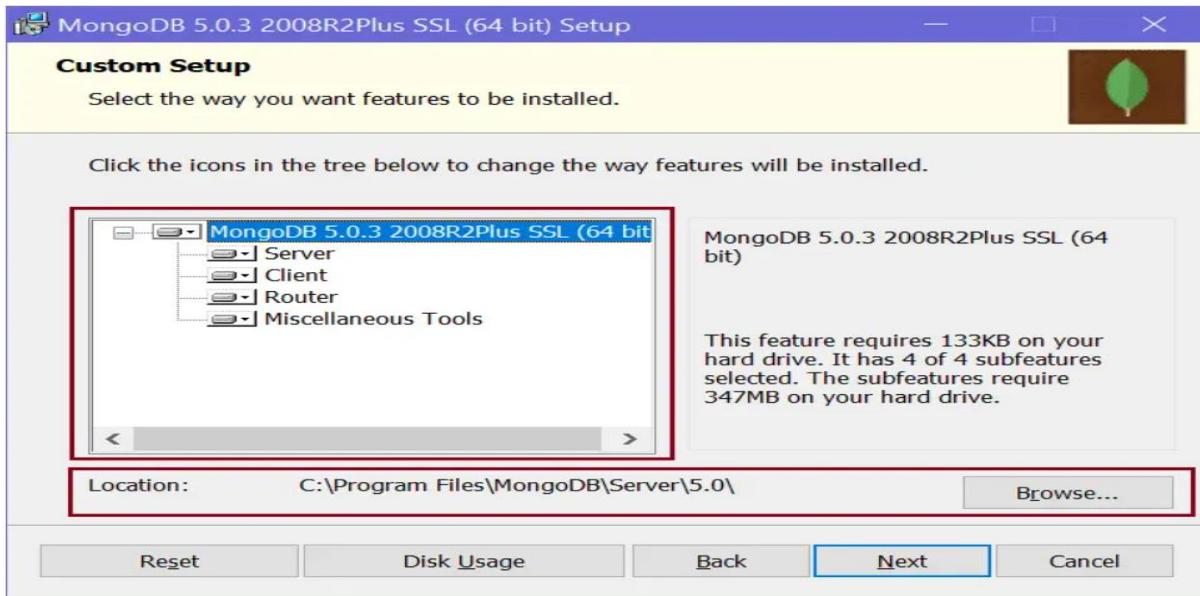
Cochez la case "J'accepte les termes du contrat de licence" et cliquez sur Suivant.



Assistant d'installation de MongoDB

Ici, vous aurez deux options d'installation : Complète et Personnalisée. L'option complète installera toutes les fonctionnalités. L'option personnalisée vous permet de sélectionner uniquement les fonctionnalités requises.

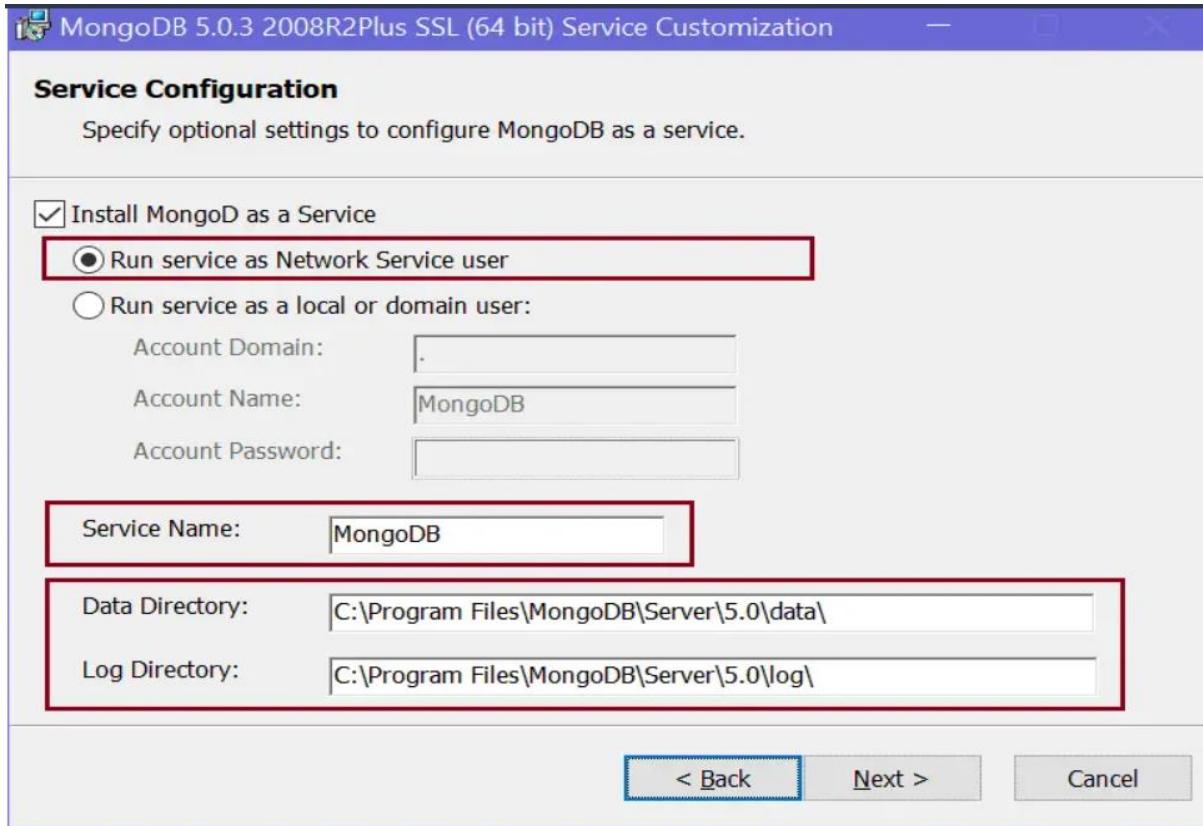
Vous pouvez sélectionner l'une de ces deux options. Ici, nous sélectionnerons l'option personnalisée juste pour vous montrer tout ce qu'elle installera. Alors, cliquez sur l'option Personnalisée qui vous amènera à l'étape suivante, comme indiqué ci-dessous.



Assistant d'installation de MongoDB

Dans la page de configuration personnalisée, développez le nœud MongoDB pour voir quelles fonctionnalités seront installées. Il installera le serveur, le client, le routeur et divers outils pour la base de données MongoDB. Il affiche également l'emplacement où MongoDB va être installé. Vous pouvez le modifier ou conserver l'emplacement par défaut et cliquer sur Suivant.

Cliquez sur Suivant pour configurer le service MongoDB, comme indiqué ci-dessous.



Configurer le service MongoDB

Le serveur MongoDB sera installé en tant que service sur votre machine Windows locale. Comme vous pouvez le voir ci-dessus, vous avez la possibilité d'exécuter un service en tant qu'utilisateur du service réseau ou en tant qu'utilisateur local ou de domaine. Nous allons sélectionner le bouton radio "Exécuter le service en tant qu'utilisateur du service réseau".

Vous pouvez modifier le nom du service par défaut mais il est recommandé de conserver le nom par défaut "MongoDB" pour l'identifier facilement.

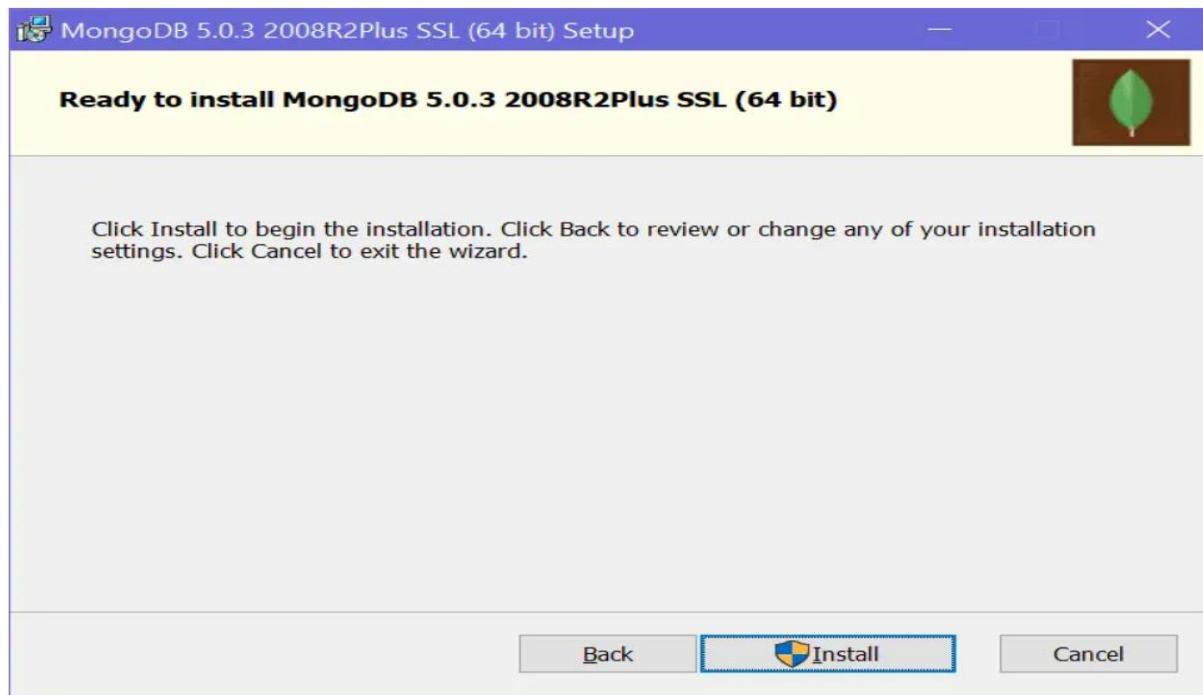
La base de données MongoDB stocke les données sous forme de fichiers BSON sur votre machine locale. Vous pouvez modifier l'emplacement par défaut où les fichiers de données et les fichiers journaux seront stockés. Ici, nous allons conserver les chemins par défaut et cliquer sur le bouton Suivant.

Sur la page suivante, cochez la case "Installer MongoDB Compass" et cliquez sur Suivant. MongoDB Compass est un outil graphique pour la base de données MongoDB où vous pouvez explorer visuellement les données, exécuter des requêtes et optimiser les performances.



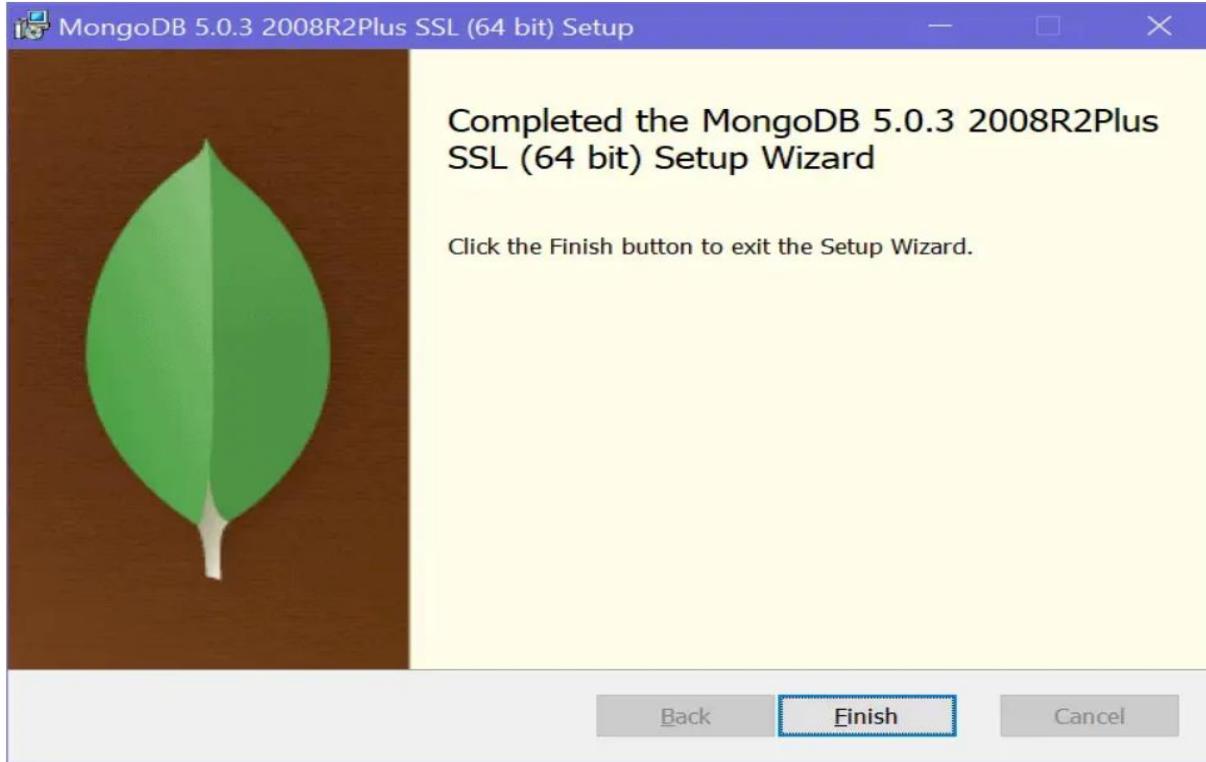
Installation de la boussole MongoDB

Ensuite, cliquez sur le bouton Installer pour lancer l'installation.



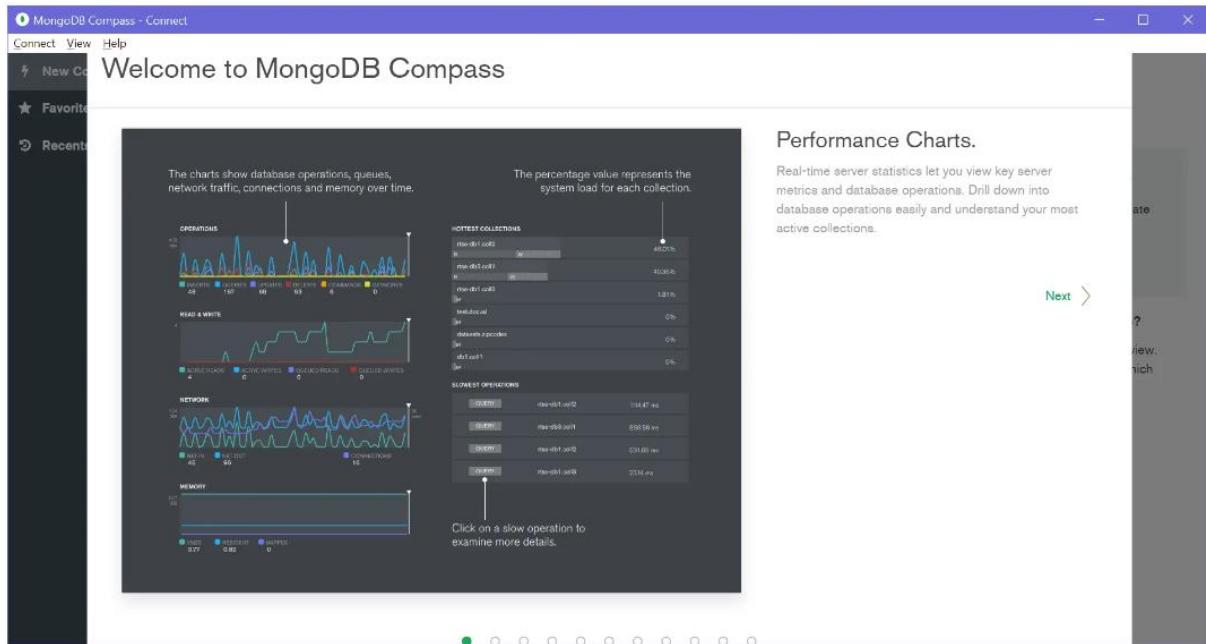
Installation de MongoDB

L'installation prendra quelques minutes. Une fois installé avec succès, cliquez sur le bouton Terminer pour fermer l'assistant.



Installation de MongoDB

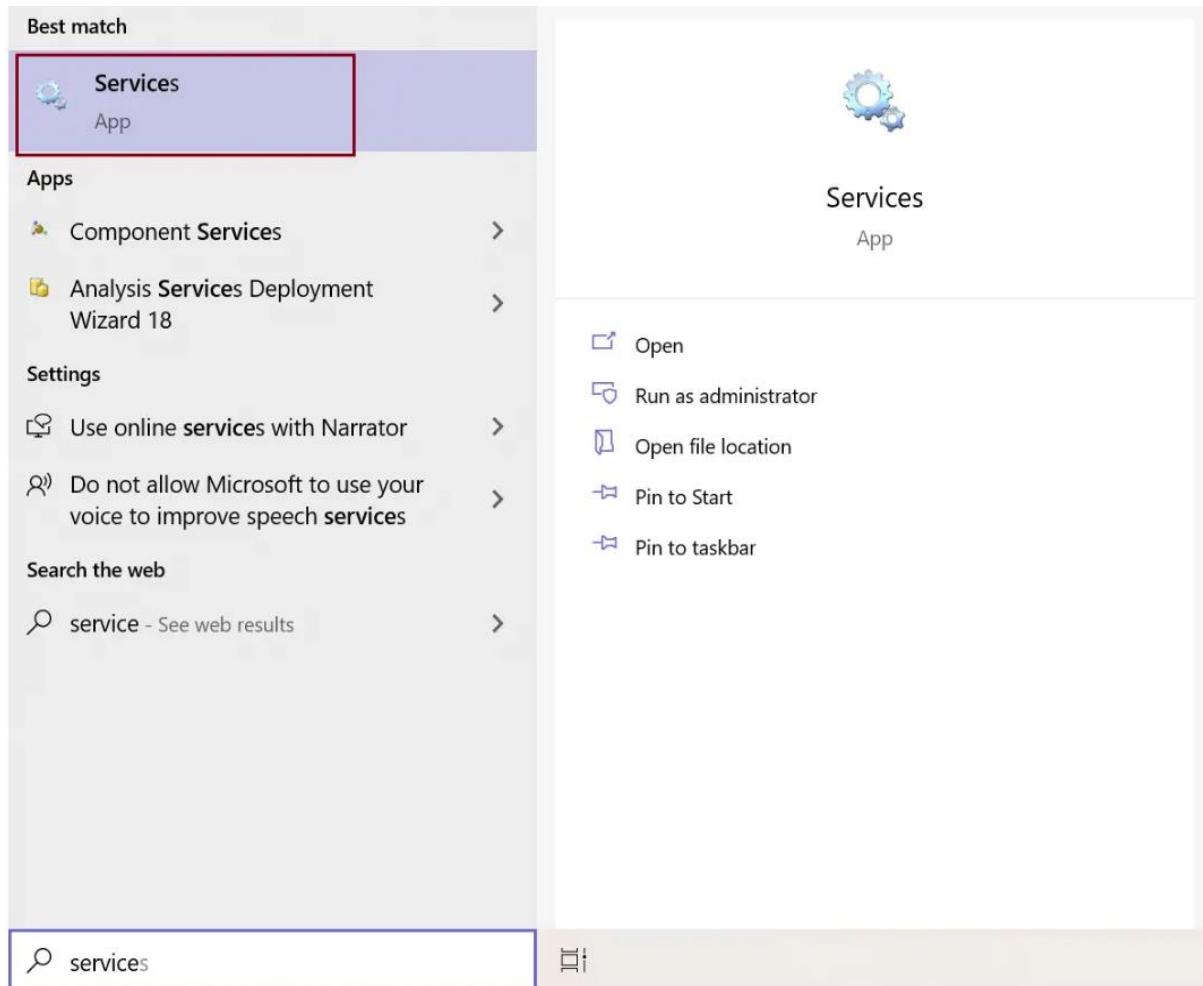
Il ouvrira également MongoDB Compass, comme indiqué ci-dessous.



Boussole MongoDB

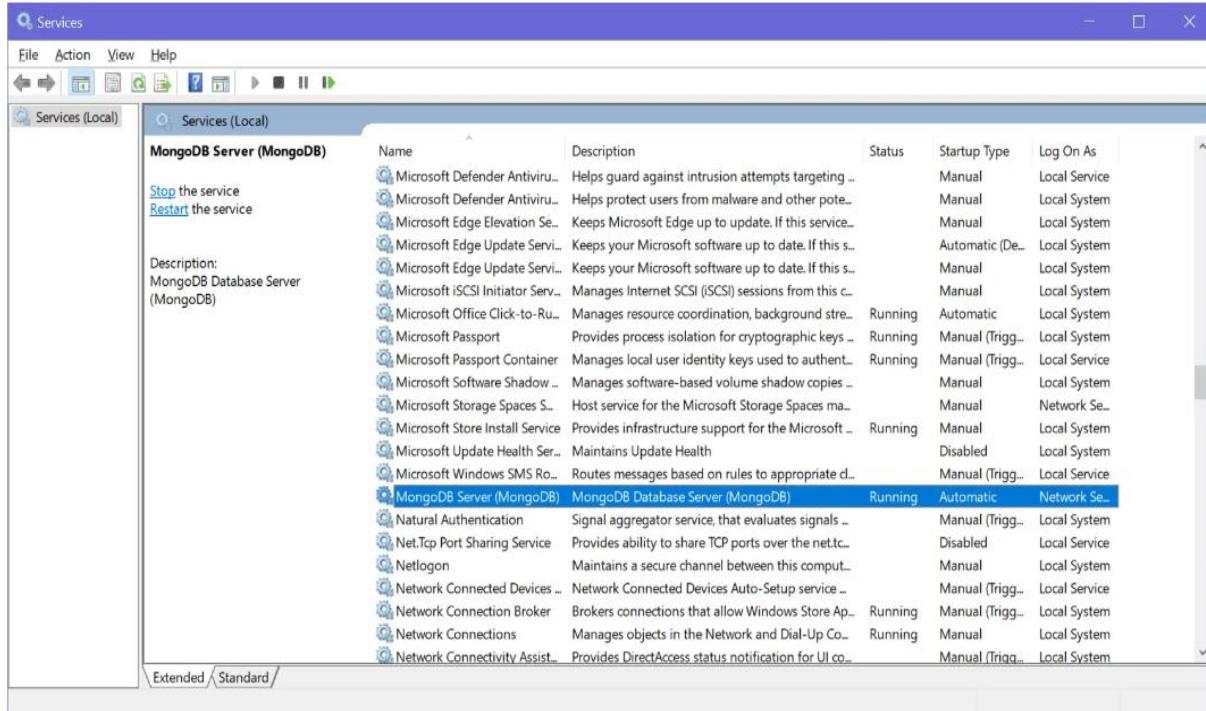
Serveur MongoDB

Nous avons installé MongoDB en tant que service réseau. Pour voir cela, ouvrez Services en recherchant "service" dans la zone de recherche Windows et cliquez sur l'application Services, comme indiqué ci-dessous.



Service MongoDB

Dans la fenêtre Services, accédez au serveur MongoDB, comme indiqué ci-dessous. Vous constaterez qu'il est déjà opérationnel.



Service MongoDB sous Windows

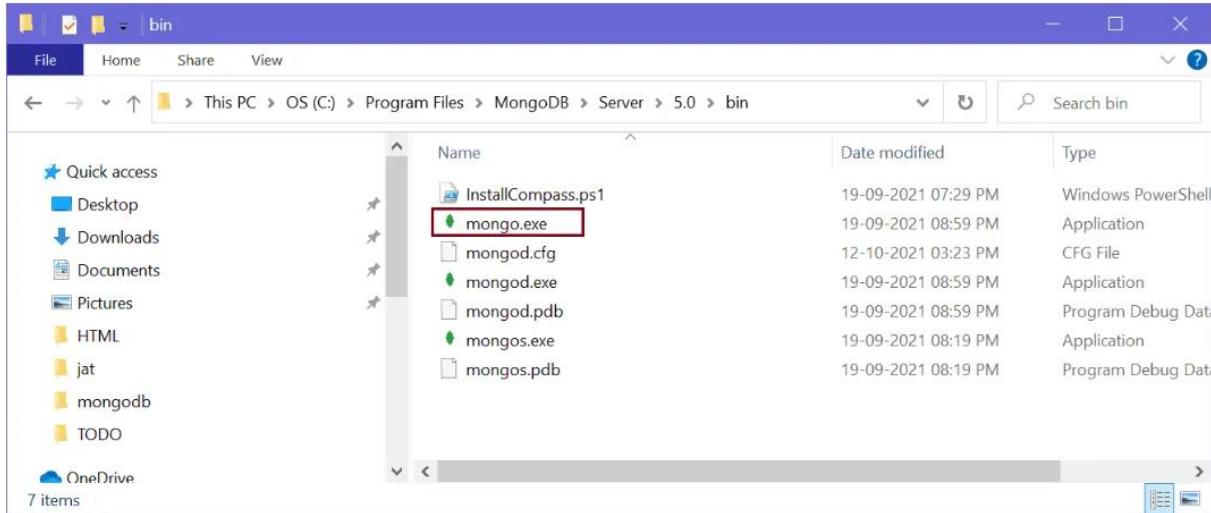
Maintenant que le service MongoDB Server est déjà en cours d'exécution, vous pouvez connecter le client MongoDB pour vous connecter à ce serveur MongoDB et exécuter les commandes.

Client MongoDB

Les clients MongoDB peuvent être votre application, MongoDB Shell, MongoDB Compass ou tout ce qui souhaite se connecter et stocker des données sur le serveur MongoDB.

Ici, nous avons déjà installé deux clients, MongoDB Shell et MongoDB Compass.

Accédez au chemin où MongoDB a été installé. Par défaut, il s'agit de "C:\Program Files\MongoDB\Server\5.0\bin", comme indiqué ci-dessous.



Dossier d'installation de MongoDB

Ici, vous trouverez mongo.exe, qui est MongoDB Shell. Cliquez sur mongo.exe pour démarrer le client, comme indiqué ci-dessous.

```
C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility. The "mongo" shell has been deprecated and will be removed in
an upcoming release.
We recommend you begin using "mongosh".
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
=====
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
https://community.mongodb.com
---
The server generated these startup warnings when booting:
2021-10-12T15:23:37.337+05:30: Access control is not enabled for the database. Read and write access to data and
configuration is unrestricted
---
---
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 to 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()
>
```

Shell MongoDB

Vous pouvez exécuter les commandes ici. Par exemple, écrivez "show dbs" et appuyez sur Entrée pour afficher la base de données dont il dispose déjà, comme indiqué ci-dessous.



```
C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
>
```

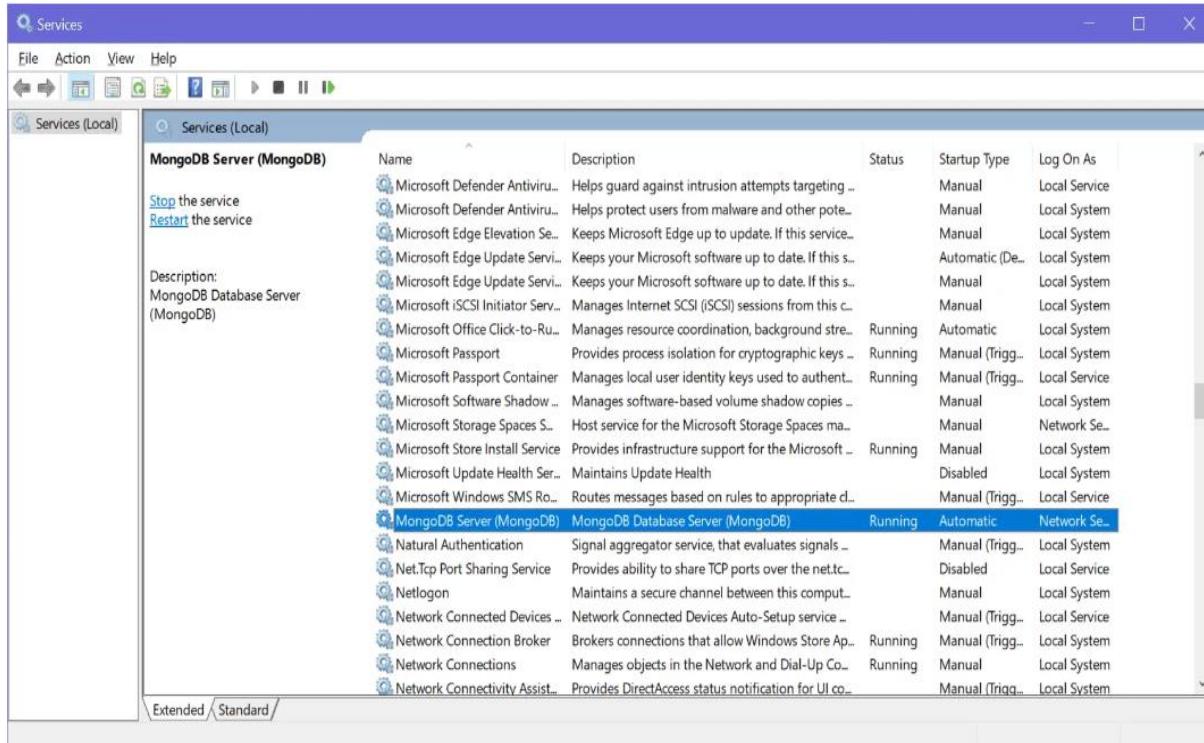
Commande Shell MongoDB

Ainsi, vous pouvez installer le serveur MongoDB, les clients (MongoDB Shell et Compass) et d'autres fonctionnalités.

Serveur MongoDB - mongod, Configurations

Ici, vous apprendrez ce qu'est le serveur MongoDB, comment l'exécuter manuellement et comment définir divers paramètres de configuration pour votre serveur MongoDB.

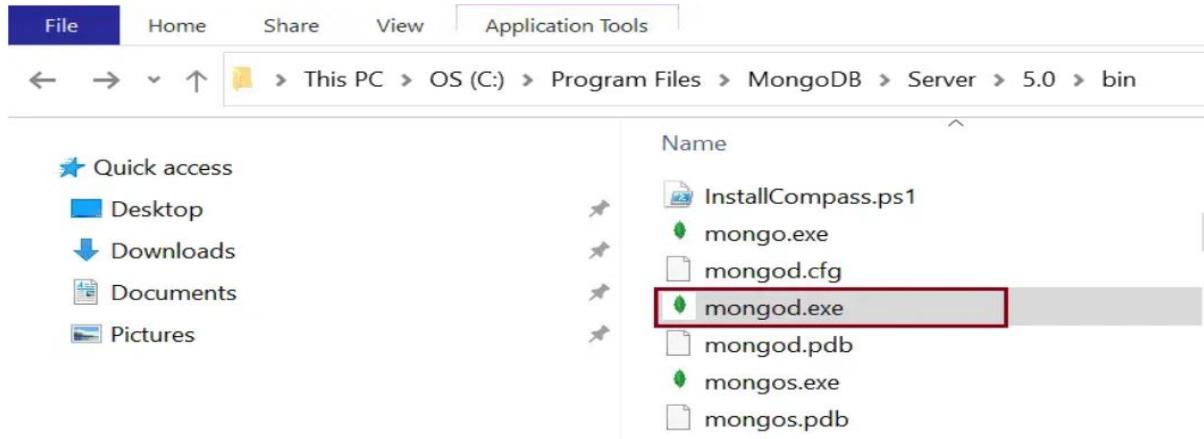
Nous avons installé le serveur MongoDB en tant que service Windows dans le chapitre [Installer MongoDB](#). Le service Windows MongoDB est automatiquement opérationnel au démarrage, comme indiqué ci-dessous.



Service MongoDB sous Windows

En interne, le service MongoDB démarre le serveur MongoDB local à l'adresse par défaut `http://127.0.0.1:27017` à chaque démarrage de votre machine. Le serveur MongoDB est

installé en tant que mongod.exe sous Windows dans le {mongodb install folder}/bin dossier, comme indiqué ci-dessous.



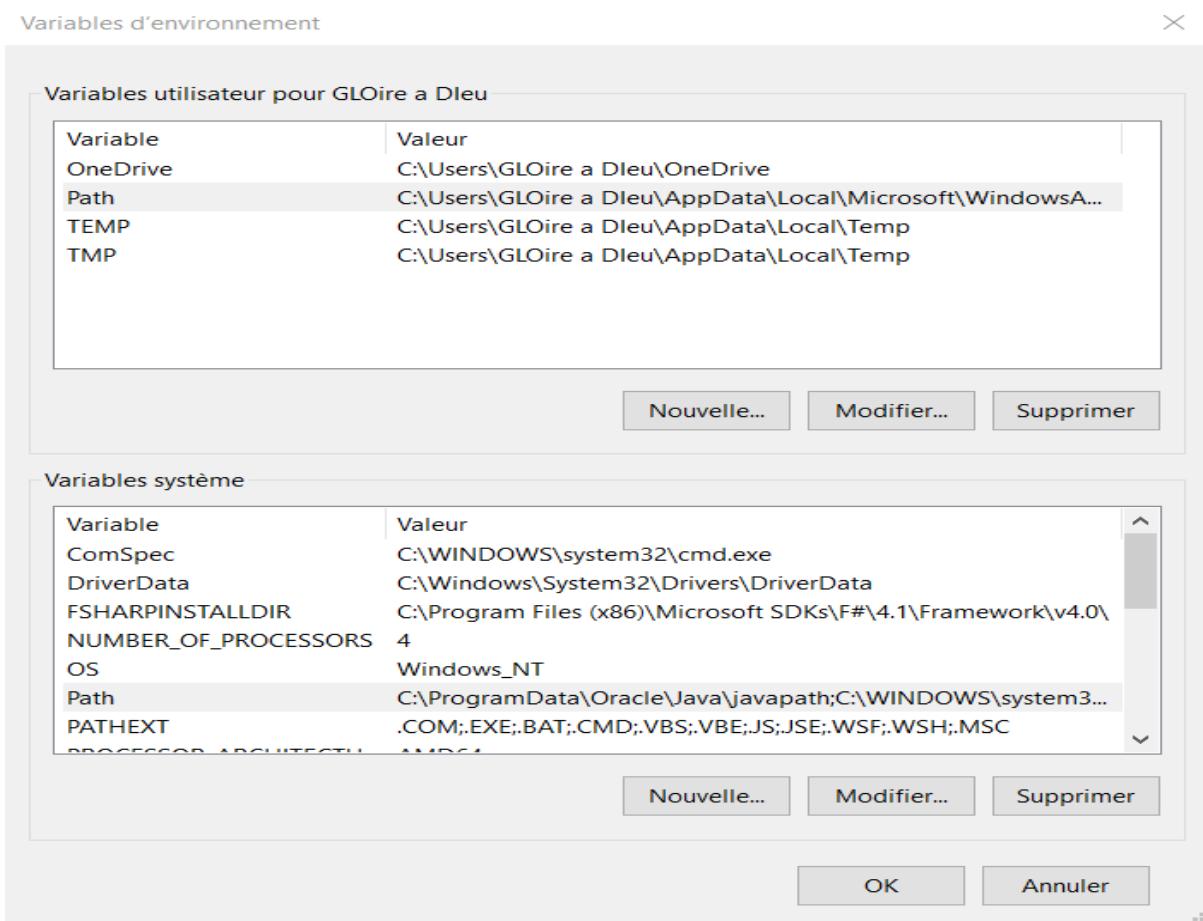
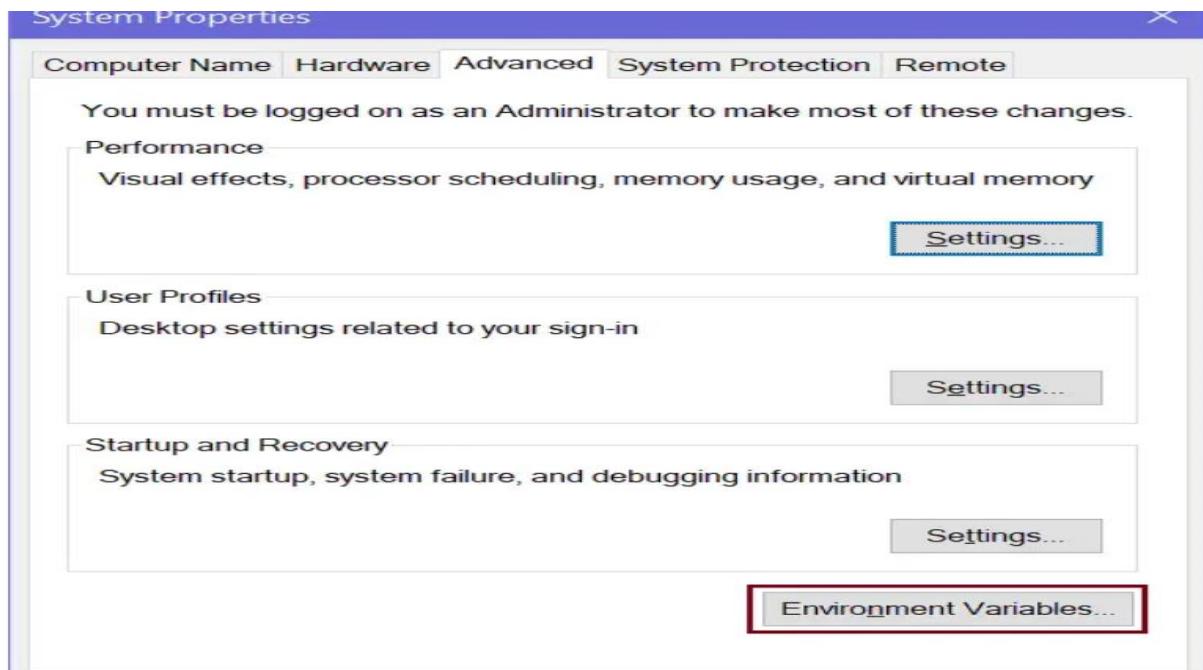
Fichier serveur MongoDB - mongod.exe

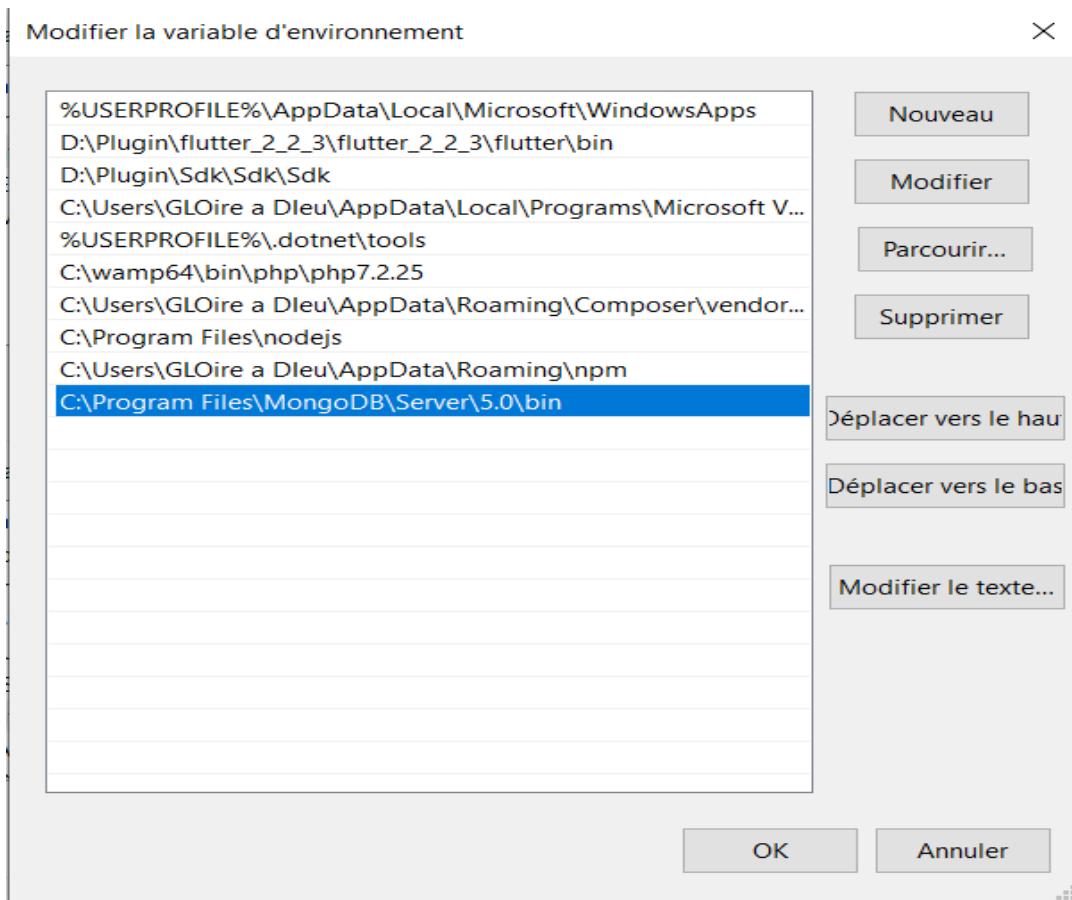
Le serveur MongoDB mongod peut être démarré ou arrêté manuellement, que vous l'ayez installé en tant que service ou non.

Exécutez mongod manuellement

Si vous avez installé MongoDB en tant que service Windows, arrêtez d'abord le service car nous allons le démarrer manuellement.

Maintenant, sélectionnez Chemin dans le volet inférieur et cliquez sur le bouton Modifier. Cela ouvrira la fenêtre "Modifier la variable d'environnement". Cliquez sur le bouton Nouveau pour ajouter le chemin du dossier MongoDB "C:\Program Files\MongoDB\Server\5.0\bin\", comme indiqué ci-dessous.

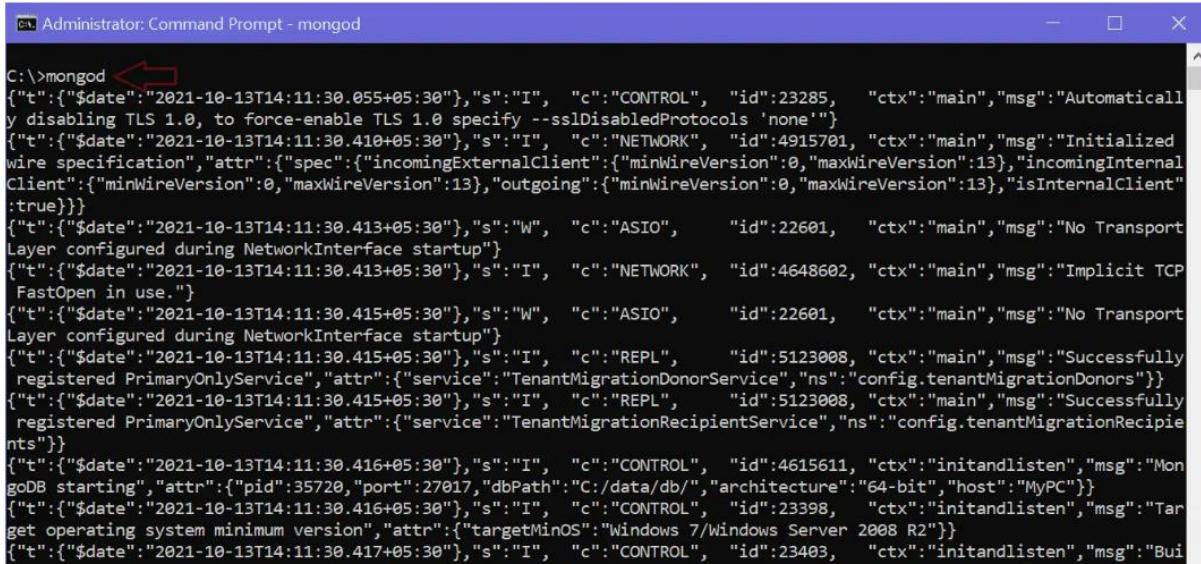




Cliquez sur OK pour fermer toutes les fenêtres contextuelles.

L'ajout du chemin du dossier MongoDB à la variable d'environnement nous permet d'exécuter mongod.exe de n'importe où sur l'invite de commande/le terminal.

Maintenant, ouvrez une invite de commande sur Windows ou un terminal en tant qu'administrateur et exécutez le mongod sur Windows ou sudo mongod sur Mac pour exécuter votre serveur MongoDB, comme indiqué ci-dessous.



```
C:\>mongod
{"t": {"$date": "2021-10-13T14:11:30.055+05:30"}, "s": "I", "c": "CONTROL", "id": 23285, "ctx": "main", "msg": "Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t": {"$date": "2021-10-13T14:11:30.410+05:30"}, "s": "I", "c": "NETWORK", "id": 4915701, "ctx": "main", "msg": "Initialized wire specification", "attr": {"spec": {"incomingExternalClient": {"minWireVersion": 0, "maxWireVersion": 13}, "incomingInternalClient": {"minWireVersion": 0, "maxWireVersion": 13}, "outgoing": {"minWireVersion": 0, "maxWireVersion": 13}, "isInternalClient": true}}}
{"t": {"$date": "2021-10-13T14:11:30.413+05:30"}, "s": "W", "c": "ASIO", "id": 22601, "ctx": "main", "msg": "No Transport Layer configured during NetworkInterface startup"}
{"t": {"$date": "2021-10-13T14:11:30.413+05:30"}, "s": "I", "c": "NETWORK", "id": 4648602, "ctx": "main", "msg": "Implicit TCP FastOpen in use."}
{"t": {"$date": "2021-10-13T14:11:30.415+05:30"}, "s": "W", "c": "ASIO", "id": 22601, "ctx": "main", "msg": "No Transport Layer configured during NetworkInterface startup"}
{"t": {"$date": "2021-10-13T14:11:30.415+05:30"}, "s": "I", "c": "REPL", "id": 5123008, "ctx": "main", "msg": "Successfully registered PrimaryOnlyService", "attr": {"service": "TenantMigrationDonorService", "ns": "config.tenantMigrationDonors"}}
{"t": {"$date": "2021-10-13T14:11:30.415+05:30"}, "s": "I", "c": "REPL", "id": 5123008, "ctx": "main", "msg": "Successfully registered PrimaryOnlyService", "attr": {"service": "TenantMigrationRecipientService", "ns": "config.tenantMigrationRecipients"}}
{"t": {"$date": "2021-10-13T14:11:30.416+05:30"}, "s": "I", "c": "CONTROL", "id": 4615611, "ctx": "initandlisten", "msg": "MongoDB starting", "attr": {"pid": 35720, "port": 27017, "dbPath": "C:/data/db/", "architecture": "64-bit", "host": "MyPC"}}
{"t": {"$date": "2021-10-13T14:11:30.416+05:30"}, "s": "I", "c": "CONTROL", "id": 23398, "ctx": "initandlisten", "msg": "Target operating system minimum version", "attr": {"targetMinOS": "Windows 7/Windows Server 2008 R2"}}
{"t": {"$date": "2021-10-13T14:11:30.417+05:30"}, "s": "I", "c": "CONTROL", "id": 23403, "ctx": "initandlisten", "msg": "Bui
```

Démarrer le serveur MongoDB

Vous pouvez spécifier diverses options de commande avec mongod command à des fins de test. Voici une liste de quelques options de commande utiles.

-h [--help] Afficher ces informations d'utilisation

--version Afficher les informations de version

-f [--config] arg Fichier de configuration spécifiant

options additionnelles

--configExpand arg Traiter les directives d'expansion dans la configuration

fichier (aucun, exec, repos)

--port arg Spécifier le numéro de port - 27017 par défaut

--ipv6 Activer le support IPv6 (désactivé par

défaut)

--listenBacklog arg (=2147483647) Définir la taille du backlog d'écoute du socket

--maxConns arg (=1000000) Nombre maximum de connexions simultanées

--pidfilepath arg Chemin d'accès complet au fichier pid (s'il n'est pas défini,

pidfile est créé)

--timeZoneInfo arg Chemin d'accès complet au répertoire d'informations sur le fuseau horaire,

par exemple /usr/share/zoneinfo

-v [--verbose] [=arg(=v)] Être plus verbeux (inclure plusieurs fois pour plus de verbosité, par exemple -vvvvv)

--quiet Sortie plus silencieuse

--logpath arg Fichier journal vers lequel envoyer l'écriture au lieu de stdout - doit être un fichier, pas annuaire

--logappend Ajouter au logpath au lieu de écraser

--logRotate arg Définir le comportement de rotation du journal (renommer|rouvrir)

--timeStampFormat arg Format souhaité pour les horodatages dans le journal messages. L'un des iso8601-utc ou iso8601-local

--setParameter arg Définir un paramètre configurable

--bind_ip arg Liste d'adresses IP séparées par des virgules écouter sur - localhost par défaut

--bind_ip_all Se lier à toutes les adresses IP

--noauth Exécuter sans sécurité

--transitionToAuth Pour la mise à niveau progressive du contrôle d'accès. Tenter de s'authentifier sur les messages sortants connexions et continuer indépendamment de succès. Accepter les connexions entrantes avec ou sans authentification.

--slowms arg (=100) Valeur de slow pour le profil et la console enregistrer

--slowOpSampleRate arg (=1) Fraction d'opérations lentes à inclure dans le profil et journal de la console

--profileFilter arg Prédicat de requête pour contrôler les opérations sont enregistrées et profilées

--auth Exécuter avec sécurité

--clusterIpSourceAllowlist arg Spécification CIDR réseau des origine pour l'accès `__system`

--profile arg 0=désactivé 1=lent, 2=tous

--cpu Afficher périodiquement le processeur et iowait utilisation

--sysinfo Affiche un système de diagnostic information

--noscripting Désactiver le moteur de script

--notablescan Ne pas autoriser les balayages de table

--keyFile arg Clé privée pour l'authentification du cluster

--clusterAuthMode arg Mode d'authentification utilisé pour le cluster authentification. Les alternatives sont (keyFile|sendKeyFile|sendX509|x509)

Options de réPLICATION :

--oplogSize arg Taille à utiliser (en Mo) pour l'opération de réPLICATION enregistrer. la valeur par défaut est de 5 % de l'espace disque (c'est-à-dire grand c'est bien)

Options d'ensemble de réPLIQUES :

--replSet arg arg est[/]
--enableMajorityReadConcern [=arg(=1)] (=1)
Active readConcern majoritaire.

enableMajorityReadConcern=false est non supporté plus longtemps

Options de partage :

--configsvr Déclare qu'il s'agit d'une base de données de configuration d'un grappe; port par défaut 27019 ; défaut répertoire /data/configdb

--shardsvr Déclare qu'il s'agit d'une base de données fragmentée d'un grappe; port par défaut 27018

Possibilités de stockage :

--storageEngine arg Quel moteur de stockage utiliser - valeurs par défaut à wiredTiger si aucun fichier de données n'est présent
--dbpath arg Répertoire pour les fichiers de données - par défaut \data\db\ qui est C:\data\db\ basé sur le lecteur de travail actuel

--directoryperdb Chaque base de données sera stockée dans un répertoire séparé

--syncdelay arg (=60) Secondes entre les synchronisations de disque

--journalCommitInterval arg (=100) la fréquence de validation de groupe/lot (ms)

--upgrade Mettre à jour la base de données si nécessaire

--repair Exécute la réparation sur toutes les bases de données

--journal Activer la journalisation

--nojournal Désactiver la journalisation (la journalisation est activée par par défaut pour 64 bits)

--oplogMinRetentionHours arg (=0) Nombre minimum d'heures à conserver dans l'oplog. La valeur par défaut est 0 (désactivé).

Les fractions sont autorisées (par exemple 1,5 heures)

Options de surveillance gratuites :

- enableFreeMonitoring arg Activer la surveillance sans cloud
(on|runtime|off)
- freeMonitoringTag arg Balises de surveillance gratuites dans le cloud

Options du gestionnaire de contrôle des services Windows :

- install Installer le service Windows
- remove Supprimer le service Windows
- reinstall Réinstaller le service Windows (équivalent
à --remove suivi de --install)
- serviceName arg Nom du service Windows
- serviceDisplayName arg Nom d'affichage du service Windows
- serviceDescription arg Description du service Windows
- serviceUser arg Compte pour l'exécution du service
- servicePassword arg Mot de passe utilisé pour s'authentifier
serviceUser

Options WiredTiger :

- wiredTigerCacheSizeGB arg Quantité maximale de mémoire à allouer
pour le cache ; Par défaut à 1/2 de physique
RAM
- zstdDefaultCompressionLevel arg (=6)
Niveau de compression par défaut pour zstandard
compresseur
- wiredTigerJournalCompressor arg (=rapide)

Utiliser un compresseur pour les enregistrements de journaux

[aucun|accrocheur|zlib|zstd]

--wiredTigerDirectoryForIndexes Placer les index et les données dans différents répertoires

--wiredTigerCollectionBlockCompressor arg (=rapide)

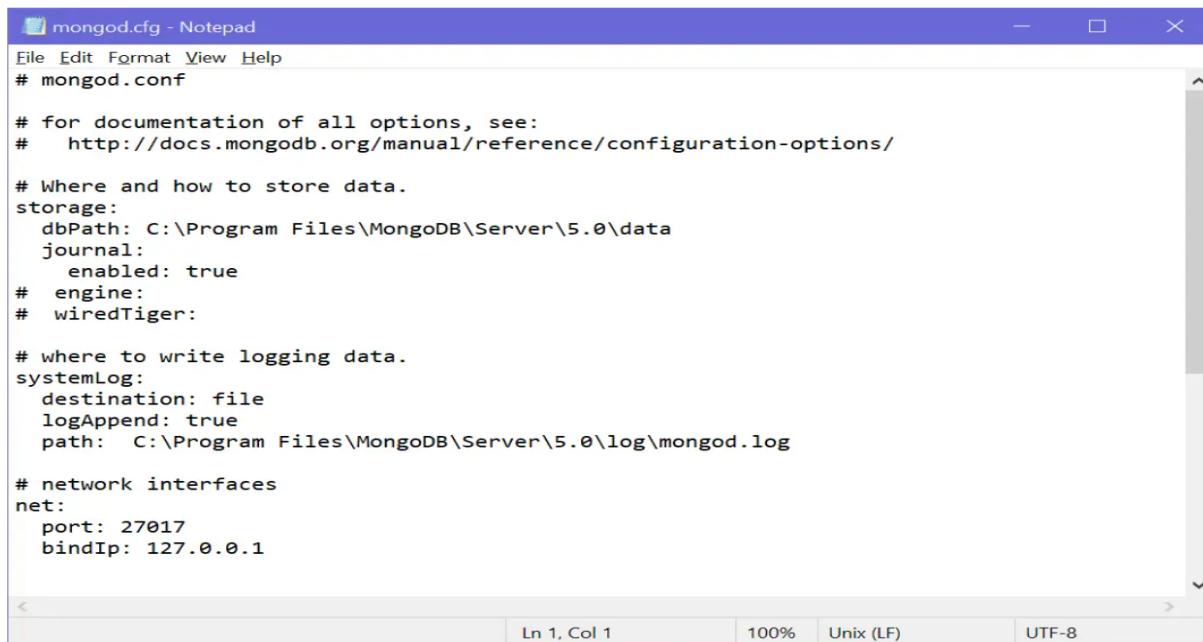
Algorithme de compression de bloc pour données de collecte [aucun|snappy|zlib|zstd]

--wiredTigerIndexPrefixCompression argument (=1)

Utiliser la compression de préfixe sur row-store feuille de vigne

Configuration du serveur MongoDB

MongoDB utilise les options du fichier de configuration pour contrôler le comportement d'une base de données. Le serveur MongoDB mongod s'exécute avec les configurations par défaut ajoutées dans le fichier de configuration. Sous Windows, il s'agit d' {install directory}\bin\mongod.cfg un fichier. Sur MacOS, c'est /usr/local/etc/mongod.conf ou /opt/homebrew/etc/mongod.conf. Et, sous Linux, c'est /etc/mongod.conf. Les fichiers de configuration MongoDB utilisent le format YAML. Voici le mongod.cfg fichier sous Windows :



```

mongod.cfg - Notepad
File Edit Format View Help
# mongod.conf

# for documentation of all options, see:
#   http://docs.mongodb.org/manual/reference/configuration-options/

# Where and how to store data.
storage:
  dbPath: C:\Program Files\MongoDB\Server\5.0\data
  journal:
    enabled: true
#  engine:
#  wiredTiger:

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: C:\Program Files\MongoDB\Server\5.0\log\mongod.log

# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1

```

The screenshot shows a Microsoft Notepad window titled "mongod.cfg - Notepad". The window contains the YAML-based configuration file for MongoDB's mongod service. The file includes sections for storage, systemLog, and net, with specific parameters like dbPath, journal.enabled, and port. The Notepad interface shows standard menu bars (File, Edit, Format, View, Help) and status bar information (Ln 1, Col 1, 100%, Unix (LF), UTF-8).

Voici l'exemple de fichier de configuration que vous aimerez peut-être utiliser pour votre serveur mongod local.

journal du système:

destinataire : fichier

chemin : "/var/log/mongodb/mongod.log"

logAppend : vrai

stockage:

journal:

activé : vrai

la gestion des processus:

fourche : vrai

filet:

bindIp : 127.0.0.1

port : 27017

setParamètre :

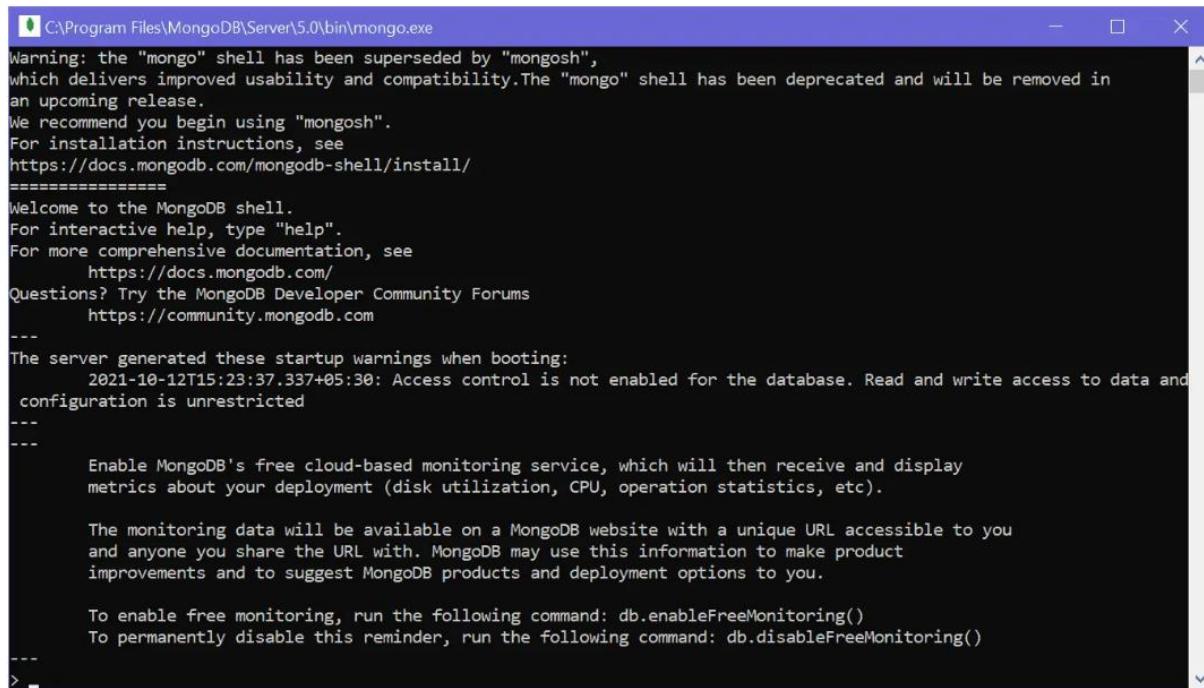
enableLocalhostAuthBypass : faux

Shell MongoDB

MongoDB Shell est le moyen le plus rapide de se connecter, de configurer, d'interroger et de travailler avec votre base de données MongoDB. Il agit comme un client de ligne de commande du serveur MongoDB.

Le shell MongoDB est un produit open source autonome et développé séparément du serveur MongoDB sous la licence Apache 2. Il s'agit d'un JavaScript entièrement fonctionnel et d'un REPL Node.js 14.x pour interagir avec les serveurs MongoDB.

MongoDB Shell est déjà installé avec MongoDB. Vous pouvez le trouver dans le répertoire d'installation où vous avez installé MongoDB. Par défaut, il s'agit de "C:\Program Files\MongoDB\Server". Ouvrez le dossier d'installation et le dossier de version approprié et accédez au dossier "bin". Ici, "mongo.exe" est le shell MongoDB. Cliquez dessus pour ouvrir le shell MongoDB, comme indiqué ci-dessous.



```
C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility. The "mongo" shell has been deprecated and will be removed in
an upcoming release.
We recommend you begin using "mongosh".
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
=====
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
    https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
    https://community.mongodb.com
---
The server generated these startup warnings when booting:
    2021-10-12T15:23:37.337+05:30: Access control is not enabled for the database. Read and write access to data and
configuration is unrestricted
---
---
    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 to 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()
---
>
```

Shell MongoDB

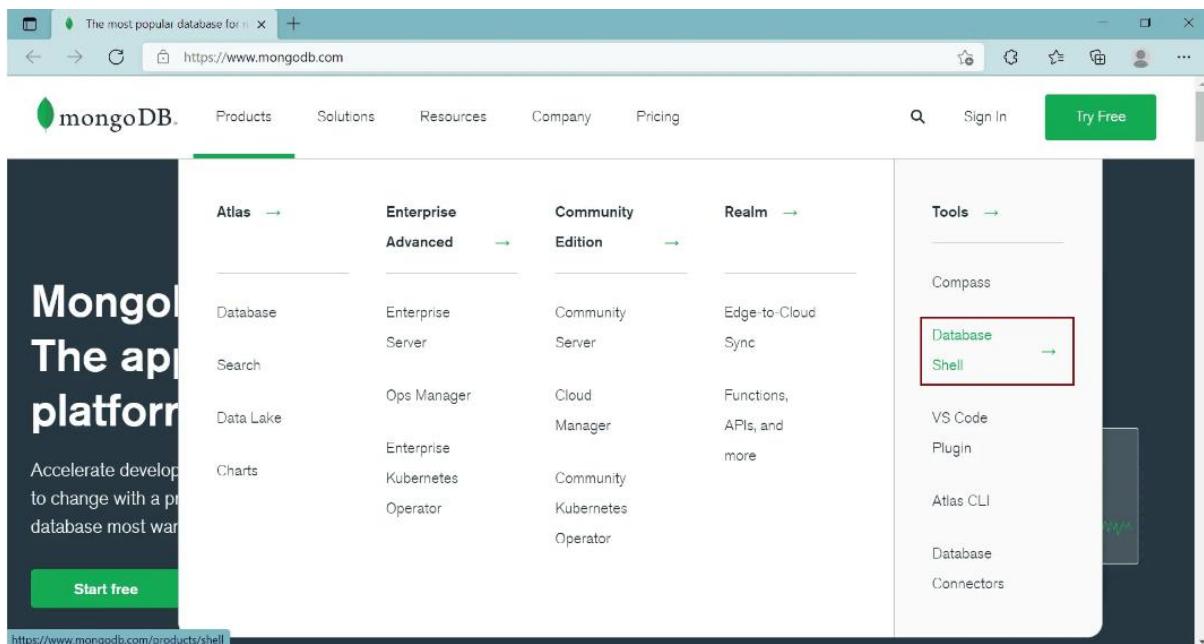
Nouveau shell MongoDB - mongosh

Dans le dossier d'installation, si vous trouvez mongosh.exe au lieu de mongo.exe, vous avez déjà un nouveau shell MongoDB. Si vous ne le trouvez pas, vous devez l'installer séparément.

Le nouveau MongoDB Shell mongosh a plus de fonctionnalités que l'ancien shell mongo, telles que la saisie semi-automatique intelligente et la coloration syntaxique, des messages d'erreur faciles à comprendre, une fonction de formatage pour présenter la sortie dans un format lisible, etc. Cependant, toutes les commandes seront exécutées en mongosh. ainsi que la coquille de mongo.

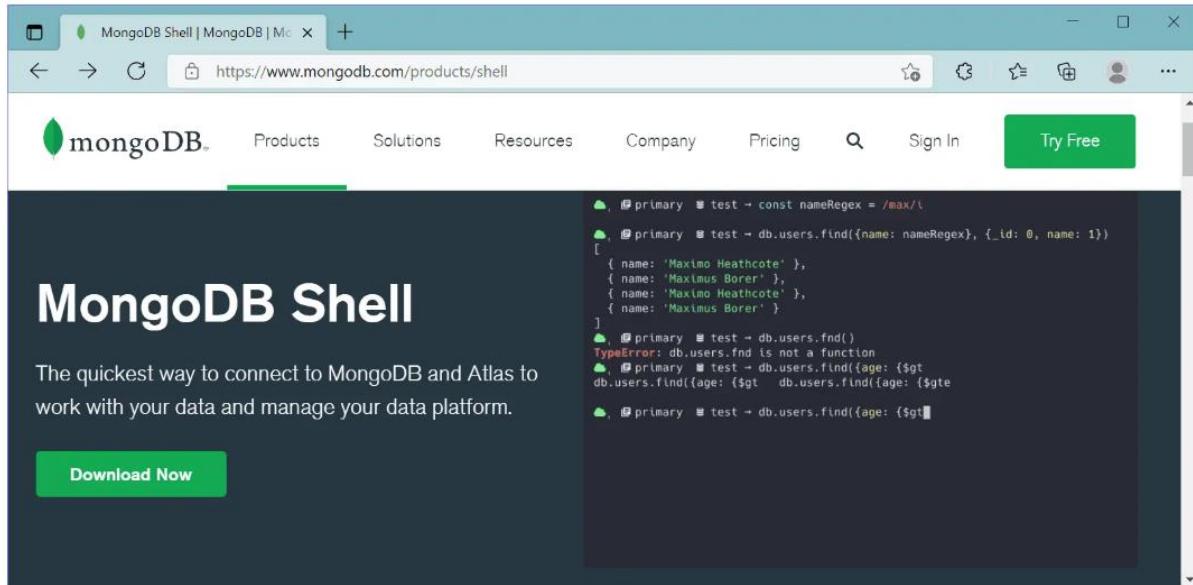
Installer mongosh

Pour installer le nouveau shell MongoDB (mongosh), visitez www.mongodb.com et cliquez sur le menu Produit -> Outils -> Shell de base de données, comme indiqué ci-dessous.



Télécharger le nouveau shell MongoDB

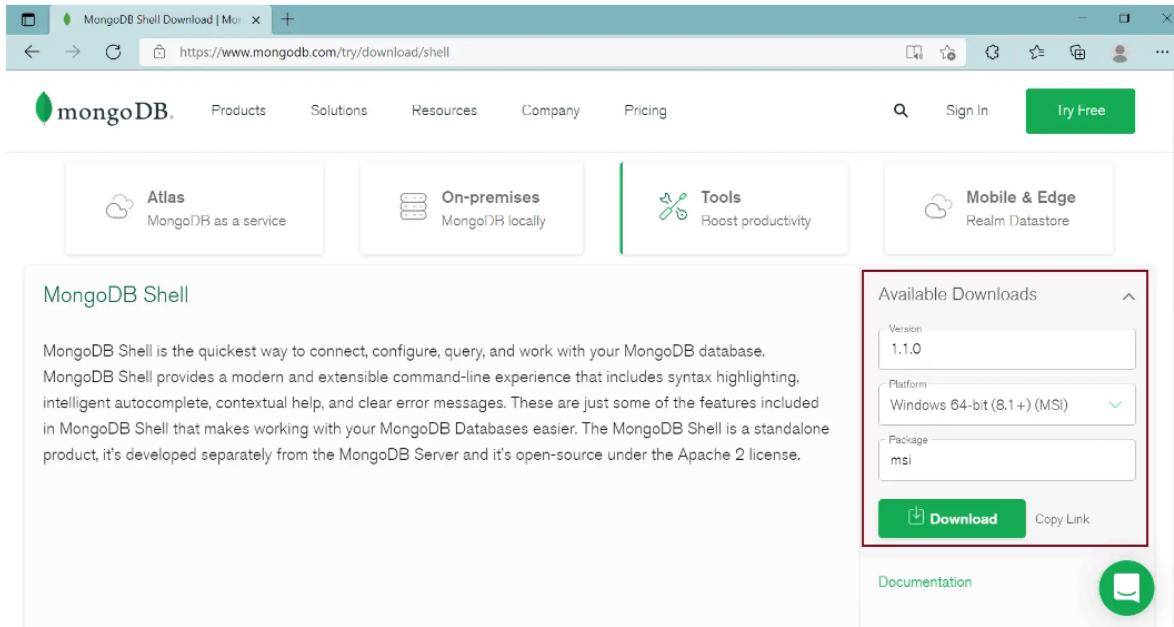
Sur la page MongoDB Shell, cliquez sur le bouton Télécharger pour télécharger le shell.



Télécharger le nouveau shell MongoDB

Cela vous amènera à une page où vous pourrez sélectionner une version, une plate-forme et un package à télécharger, comme indiqué ci-dessous.

Cela vous amènera à une page où vous pourrez sélectionner une version, une plate-forme et un package à télécharger, comme indiqué ci-dessous.

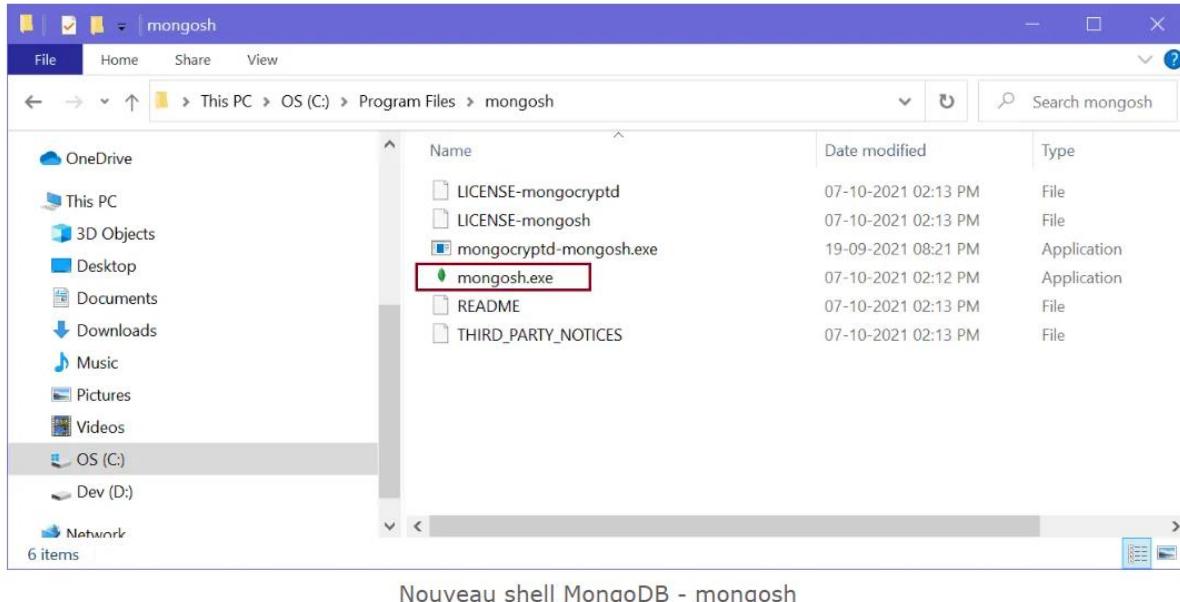


Télécharger le nouveau shell MongoDB

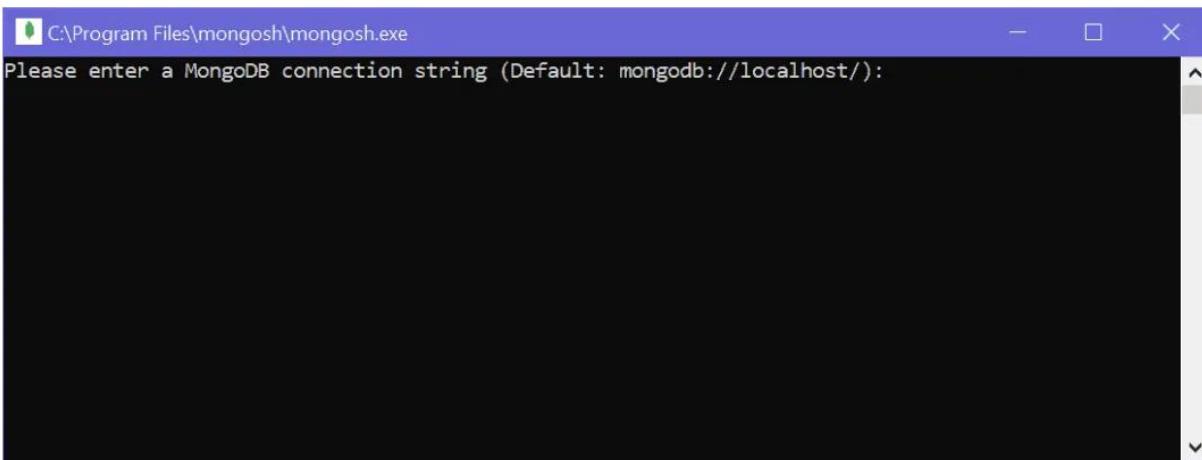
Cliquez sur le bouton Télécharger pour télécharger le fichier d'installation.

Maintenant, cliquez sur le fichier d'installation téléchargé pour démarrer l'assistant d'installation.

Cela devrait avoir installé mongosh dans le dossier "C:\Program Files\mongosh" sous Windows, comme indiqué ci-dessous.

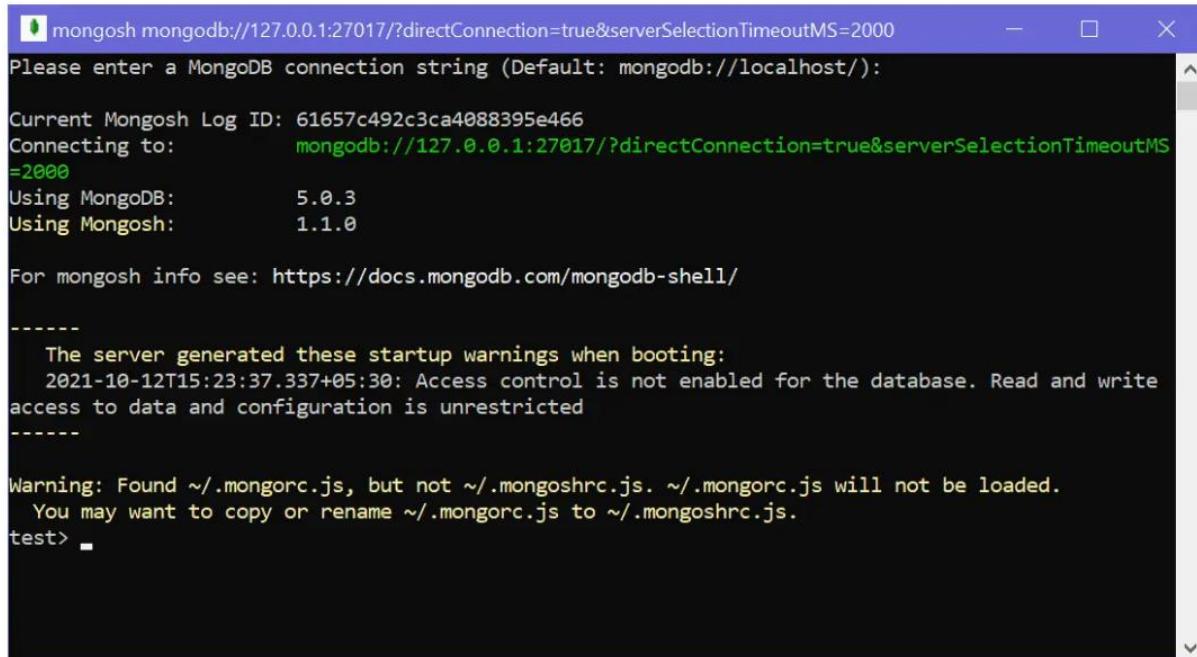


Cliquez sur mongosh.exe pour ouvrir un nouveau shell MongoDB, comme indiqué ci-dessous.



Nouveau shell MongoDB - mongosh

Appuyez sur Entrée pour démarrer le shell, comme indiqué ci-dessous.



```

mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Please enter a MongoDB connection string (Default: mongodb://localhost/):

Current Mongosh Log ID: 61657c492c3ca4088395e466
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Using MongoDB:      5.0.3
Using Mongosh:      1.1.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

-----
The server generated these startup warnings when booting:
2021-10-12T15:23:37.337+05:30: Access control is not enabled for the database. Read and write
access to data and configuration is unrestricted
-----

Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be loaded.
You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.
test> -

```

Nouveau shell MongoDB - mongosh

Sinon, ouvrez une nouvelle invite de commande sous Windows et écrivez mongosh et appuyez sur Entrée. Il ouvrira le même shell MongoDB.

Exécuter les commandes MongoDB

Vous pouvez exécuter des commandes MongoDB pour les opérations CRUD sur le shell MongoDB (mongo ou mongosh). Par exemple, exécutez la commande "show dbs" pour voir toutes les bases de données sur le serveur MongoDB connecté.

```
> show dbs
admin 41 ko
config 111 ko
local 41 ko
```

Utilisez la commande "db" pour vérifier la base de données actuelle.

```
>
test de base de données
```

Exécutez la .editorcommande pour exécuter des commandes multilignes. Appuyez sur Ctrl + dpour exécuter une commande ou Ctrl + csur pour annuler.

Le shell MongoDB est JavaScript et Node.js REPL, vous pouvez donc également exécuter du code JavaScript limité.

```
> "Bonjour".longueur
5
```

Commandes du shell MongoDB

MongoDB Shell est le moyen le plus rapide de se connecter, de configurer, d'interroger et de travailler avec votre base de données MongoDB. Il agit comme un client de ligne de commande du serveur MongoDB.

Vous pouvez démarrer MongoDB Shell en exécutant mongoor mongoshcommand sur l'invite de commande/terminal. mongoshest le nouveau shell MongoDB avec plus de fonctionnalités que l'ancien mongoshell.

```
mongosh <commandes>
```

La --helpcommande affiche toutes les commandes que vous pouvez utiliser avec mongoou mongosh, comme indiqué ci-dessous.

Par exemple, le --nodbvous permet d'exécuter le shell MongoDB sans vous connecter à une base de données.

```
C:\>mongosh --nodb
ID de journal Mongosh actuel : 6166bab1a1acde9f7f388c27
Utilisation de Mongosh : 1.1.0

Pour les informations mongosh, voir : https://docs.mongodb.com/mongodb-shell/

Avertissement : Trouvé ~/.mongorc.js, mais pas ~/.mongoshrc.js. ~/.mongorc.js ne sera pas chargé
Vous pouvez copier ou renommer ~/.mongorc.js en ~/.mongoshrc.js.
>
```

Connectez-vous à la base de données MongoDB

Par défaut, la commande mongosh ou mongo connecte à la base de données MongoDB locale sur le localhost:27017. Donc, mongosh et mongo "mongodb://localhost:27017" sont les mêmes qui se connectent à une base de données sur l'hôte local au port 27017.

Pour vous connecter à la base de données locale sur un autre port, utilisez l' --port option :

```
C:\>mongosh --port 23023
```

Ce qui suit se connecte à la base de données distante sur mymongodb.example.com le port 23023.

```
C:\>mongosh "mongodb://my.mongodb.example.com:23023"
```

Ou, utilisez les options --host et --port:

```
mongosh --host mongodb0.example.com --port 28015
```

Utilisez les options de ligne de commande --username et --authenticationDatabase pour vous connecter à la base de données nécessitant une authentification.

```
mongosh "mongodb://my.mongodb.example.com:23023" --username steve --authenticationDatabase admin
```

En savoir plus sur les options [de connexion à mongodb](#).

Aide sur la commande MongoDB

Vous pouvez également obtenir de l'aide sur les commandes mongosh ou mongo après vous être connecté à une base de données à l'aide de la help commande.

Comme vous pouvez le voir ci-dessus, utilisez diverses commandes pour travailler avec la base de données connectée. Ajoutez .help la commande ci-dessus pour obtenir l'aide de cette commande. Par exemple, exécutez la show.help commande pour afficher l'aide de show, comme indiqué ci-dessous.

```
test> show.help
```

'show databases'/'show dbs' : affiche une liste de toutes les bases de données disponibles.

'show collections'/'show tables' : affiche une liste de toutes les collections de la base de données actuelle.

'show profile' : imprime les informations de system.profile.

'show users' : imprime une liste de tous les utilisateurs de la base de données actuelle.

'show roles' : affiche une liste de tous les rôles pour la base de données actuelle.

'afficher le journal' : journal pour la connexion actuelle, si le type n'est pas défini, utilise 'global'
'show logs' : imprime tous les journaux.

La show dbscommande affichera toutes les bases de données sur le serveur connecté.

tester> afficher la base de données

administrateur 41 Ko

configuration 73,7 ko

ressources humainesb 41 kB

locale 73,7 ko

Les show collectionscommandes montrent les collections dans la base de données actuelle admin.

admin> afficher les collections

système.version

Utilisez la db.help()commande pour obtenir l'aide sur la commande db.

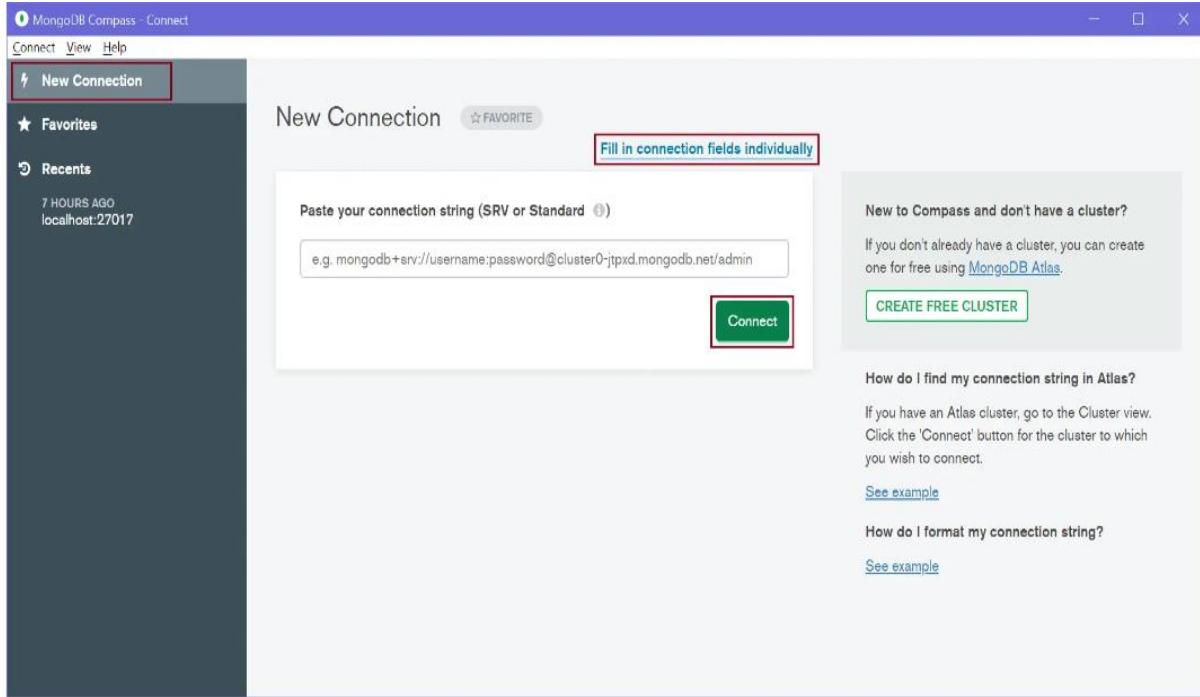
Utilisez db.test.help()la commande pour obtenir de l'aide sur les commandes liées à la collection.

Boussole MongoDB

MongoDB Compass est un outil basé sur une interface graphique (unline MongoDB Shell) pour interagir avec le serveur et les bases de données MongoDB locaux ou distants. Utilisez Compass pour explorer visuellement vos données, exécuter des requêtes ad hoc, effectuer des opérations CRUD et afficher et optimiser les performances de vos requêtes. Il peut être installé sur Linux, Mac ou Windows.

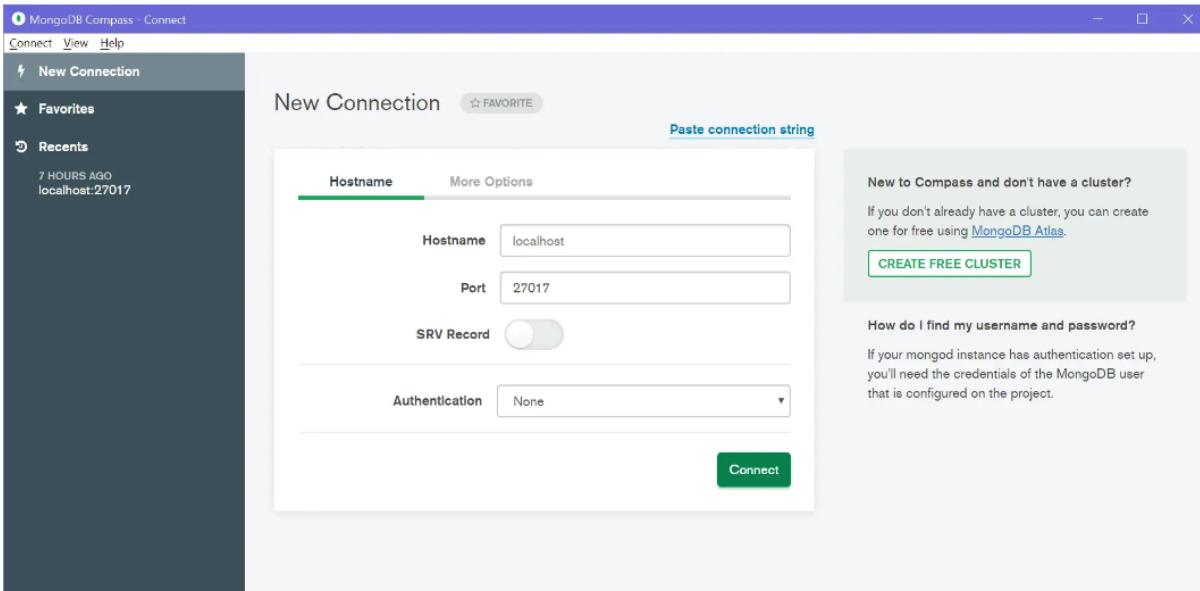
Si vous n'avez pas installé MongoDB Compass avec MongoDB, téléchargez la version communautaire gratuite de Compass pour votre plateforme.

Maintenant, effectuez une recherche de fenêtre sur "mongodb compass" et ouvrez-la, comme indiqué ci-dessous.



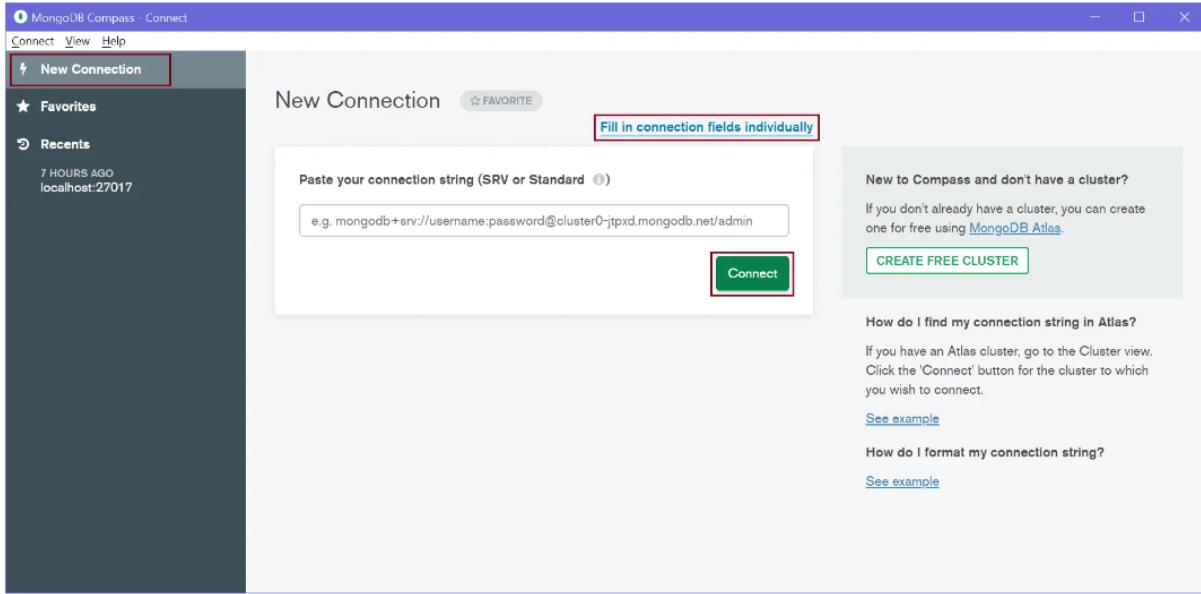
Boussole MongoDB

Sur la page principale, cliquez sur l' New Connectiononglet dans le volet de gauche pour ouvrir New Connectionla page ci-dessus. Vous pouvez coller une chaîne de connexion ou cliquer sur Fill in connection fields individuallyle lien. Cela ouvrira la fenêtre suivante où vous pourrez entrer le nom d'hôte, le port, le nom d'utilisateur, le mot de passe, etc.



Boussole MongoDB

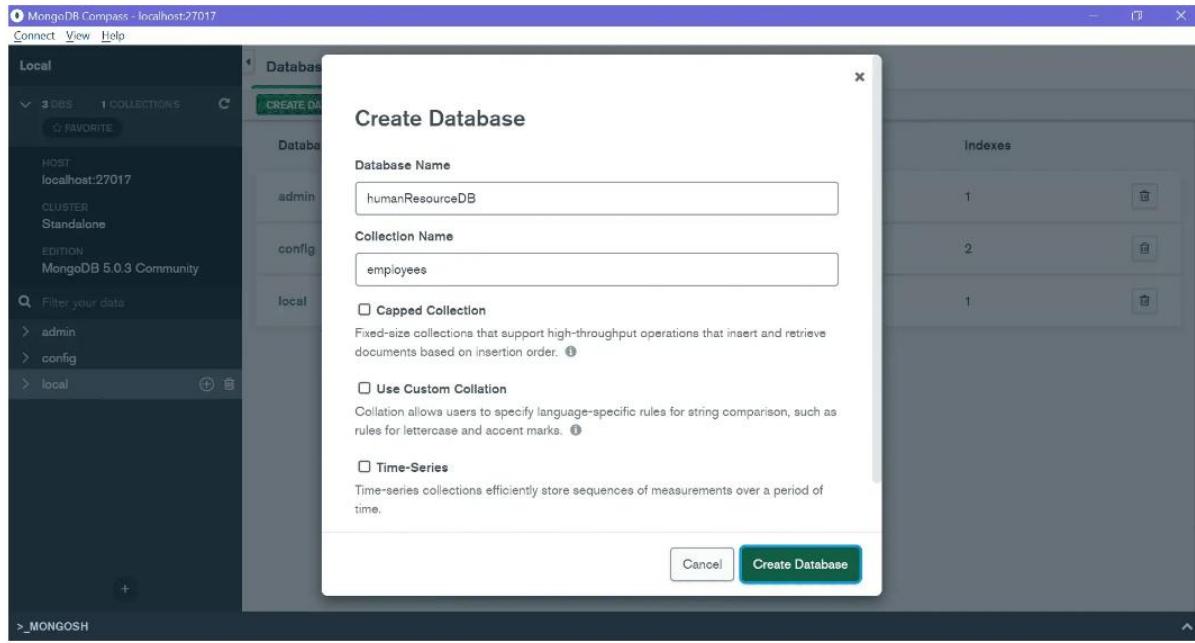
Cela a déjà des paramètres par défaut, cliquez simplement sur OK pour vous connecter à votre serveur local, comme indiqué ci-dessous.



Boussole MongoDB

Comme vous pouvez le voir ci-dessus, il affichera toutes les bases de données sur le serveur MongoDB connecté. Dans le volet de gauche, il affiche des informations sur le serveur connecté.

Désormais, vous pouvez créer, modifier, supprimer des bases de données, des collections, des documents à l'aide de MongoDB Compass. Cliquez sur le CREATE DATABASE bouton pour créer une nouvelle base de données. Cela ouvrira Create Database une fenêtre contextuelle, comme indiqué ci-dessous.



MongoDB Compass - Créer une base de données

Saisissez le nom de votre base de données et le nom de votre collection, puis cliquez sur Create Database. Cela créera une nouvelle base de données humanResourceDB avec la nouvelle employees collection présentée ci-dessous.

Boussole MongoDB - Collections

Cliquez sur employees la collection pour y insérer, mettre à jour, trouver des documents. Cela ouvrira la fenêtre suivante pour gérer les documents.

This collection has no data

It only takes a few seconds to import data from a JSON or CSV file

Import Data

MongoDB Compass - Gérer les documents

CHAPITRE 2 : CREATETION DES BASES DES DONNEES, DES COLLECTIONS ET DES DOCUMENTS

MongoDB est une base de données NoSQL open source orientée document. C'est l'une des bases de données NoSQL les plus populaires et les plus utilisées. Dans ce didacticiel, vous apprendrez à créer une nouvelle base de données MongoDB ou à passer à une base de données existante.

Une base de données est un endroit où les données sont stockées de manière organisée. Dans MongoDB, les bases de données sont utilisées pour stocker des collections. Un seul serveur MongoDB peut avoir plusieurs bases de données et une seule base de données MongoDB peut avoir plusieurs collections.

Vous pouvez utiliser [MongoDB Shell](#) ou [MongoDB Compass](#) pour créer une nouvelle base de données.

MongoDB fournit la `use <database-name>` commande pour se connecter à la base de données. Si le nom de base de données spécifié n'existe pas, il le crée et le définit comme base de données actuelle.

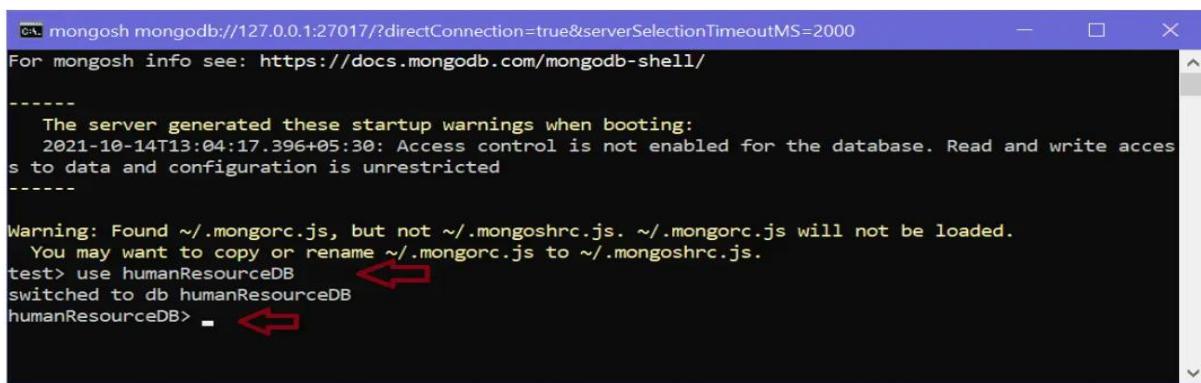
Par exemple, la commande suivante passe à la base de données "humanResouredb". S'il n'existe pas, il le crée.

Exemple : changer ou créer une base de données

Copie

`use humanResourceDB`

Ce qui suit montre comment créer ou changer de base de données MongoDB dans le shell MongoDB mongosh :



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
For mongosh info see: https://docs.mongodb.com/mongodb-shell/
-----
The server generated these startup warnings when booting:
2021-10-14T13:04:17.396+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----
Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be loaded.
You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.
test> use humanResourceDB
switched to db humanResourceDB
humanResourceDB>
```

Créer ou changer de base de données dans MongoDB Shell

Comme vous pouvez le voir ci-dessus, "admin", "config" et "local" sont des bases de données par défaut. A partir de maintenant, "humanResourceDB" n'est pas visible. C'est parce qu'il n'y a pas de collection.

Pour supprimer une base de données, utilisez la `db.dropDatabase()` méthode qui supprime une base de données courante.



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
humanResourceDB> db.dropDatabase()
{ ok: 1, dropped: 'humanResourceDB' }
humanResourceDB>
```

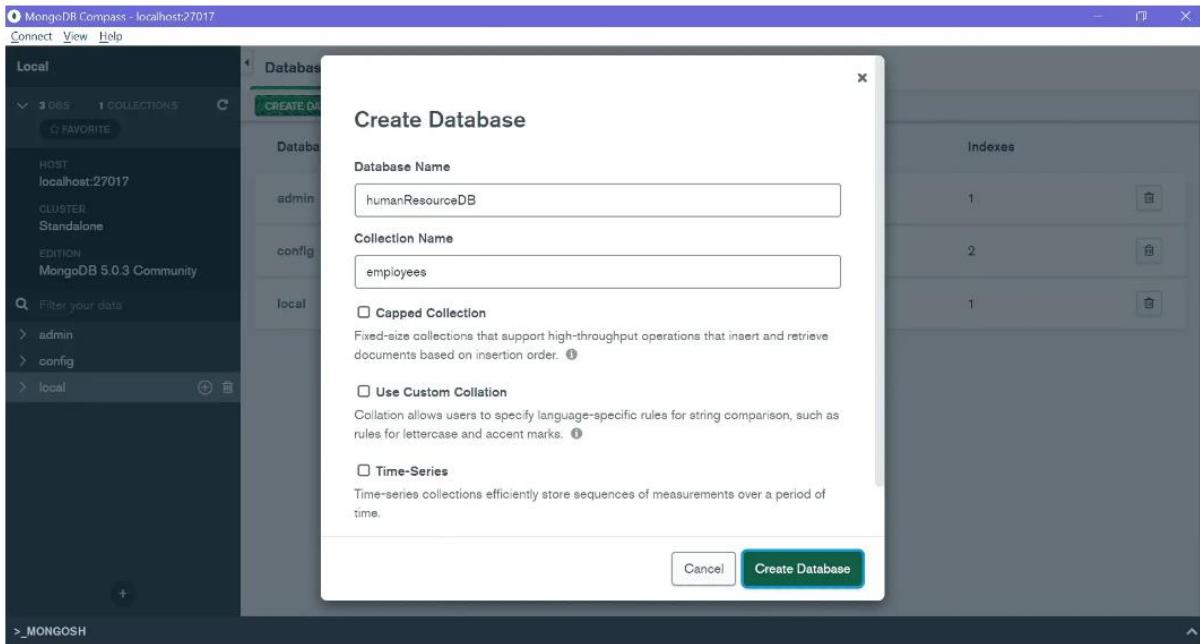
Supprimer la base de données

Ci-dessus, `{ ok: 1, dropped: 'humanResourceDB' }` indique que la base de données a été supprimée avec succès.

Remarque : Les noms de méthode sont sensibles à la casse. Ainsi, l'exécution `db.dropdatabase()` générera une erreur.

Créer une base de données à l'aide de MongoDB Compass

Vous pouvez créer une nouvelle base de données à l'aide de MongoDB Compass. Pour cela, ouvrez Compass et connectez-vous à votre base de données locale ou distante. Une fois qu'il se connecte au serveur MongoDB, cliquez sur le bouton supérieur "CRÉER UNE BASE DE DONNÉES" qui ouvrira la fenêtre contextuelle, comme indiqué ci-dessous.



MongoDB Compass - Créer une base de données

Saisissez le nom de votre base de données et le nom de votre collection, puis cliquez sur **Create Database**. Cela créera une nouvelle base de données `humanResourceDB` avec la nouvelle collection `employees` présentée ci-dessous.

The screenshot shows the MongoDB Compass interface with the 'humanResourceDB' database selected. The left sidebar shows the database structure: 'admin', 'config', 'local', and 'humanResourceDB' (which contains 'employees'). The main area is titled 'Collections' and shows a table with one row for 'employees'. The table columns are: Collection Name, Documents, Avg. Document Size, Total Document Size, Num. Indexes, Total Index Size, and Properties. The 'employees' row has 0 documents, 0.0 B total size, 1 index, and 4.0 KB total index size. The 'CREATE COLLECTION' button is highlighted with a red box.

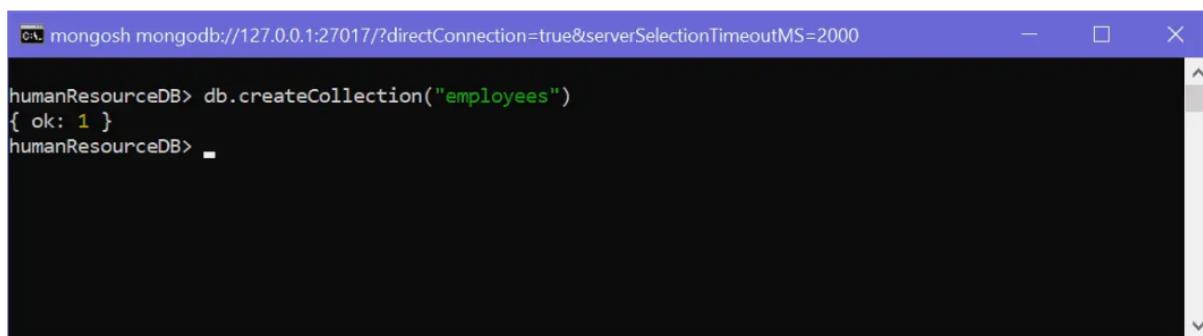
Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
employees	0	-	0.0 B	1	4.0 KB	

Boussole MongoDB - Collections

Collections MongoDB

Une collection dans MongoDB est similaire à une table dans RDBMS. Les collections MongoDB n'appliquent pas les schémas. Chaque collection MongoDB peut avoir plusieurs documents. Un document équivaut à ligne dans une table dans RDBMS.

Pour créer une collection, utilisez la `db.createCollection()` commande. Ce qui suit crée une nouvelle `employees` collection dans la base de données actuelle, qui est `humanResourceDB` la base de données créée dans le chapitre précédent.

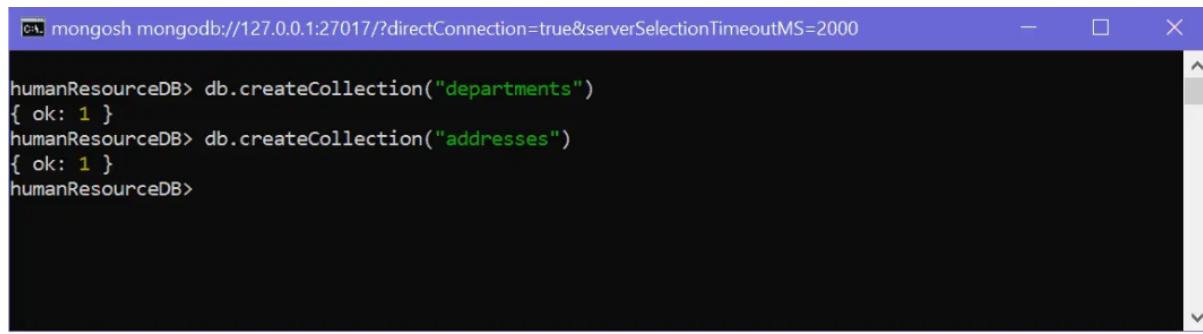


```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
humanResourceDB> db.createCollection("employees")
{ ok: 1 }
humanResourceDB>
```

Créer une collection

Ci-dessus, la `employees` collection est créée à l'aide de la `creatCollection()` méthode. Elle renvoie un objet `{ ok: 1 }`, qui indique que la collection a été créée avec succès.

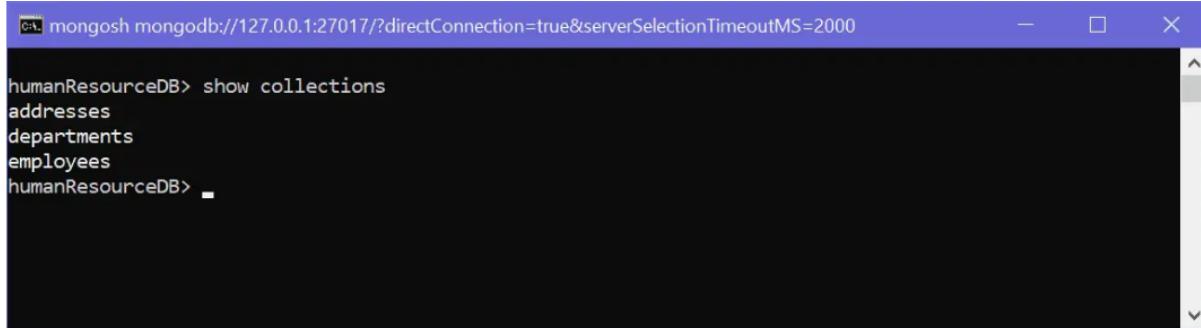
Comme mentionné ci-dessus, une seule base de données peut avoir plusieurs collections. Ce qui suit crée plusieurs collections.



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
humanResourceDB> db.createCollection("departments")
{ ok: 1 }
humanResourceDB> db.createCollection("addresses")
{ ok: 1 }
humanResourceDB>
```

Créer plusieurs collections

Utilisez les `show collections` commandes pour répertorier toutes les collections d'une base de données.



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
humanResourceDB> show collections
addresses
departments
employees
humanResourceDB>
```

Afficher les collections

Pour supprimer une collection, utilisez la `db.<collection-name>.drop()` méthode.



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
humanResourceDB> db.addresses.drop() ←
true
humanResourceDB> show collections ←
departments
employees
humanResourceDB>
```

Supprimer la collection

Créer une collection dans MongoDB Compass

Pour créer une nouvelle collection à l'aide de MongoDB Compass, connectez Compass à votre serveur et sélectionnez la base de données.

Cliquez sur le bouton "Créer une collection" pour créer une nouvelle collection, comme indiqué ci-dessous.

MongoDB Compass - localhost:27017/humanResourceDB

Connect View Help

Local

HOST localhost:27017

CLUSTER Standalone

EDITION MongoDB 5.0.3 Community

Filter your data

- > admin
- > config
- < humanResourceDB
 - employees
- > local

Collections

CREATE COLLECTION

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
employees	0	-	0.0 B	1	4.0 KB	

Boussole MongoDB - Collections

Entrez le nom d'une collection, cochez la case appropriée et cliquez sur le **Create Collection** bouton pour la créer.

MongoDB Compass - localhost:27017/humanResourceDB

Connect View Help

Local

HOST localhost:27017

CLUSTER Standalone

EDITION MongoDB 5.0.3 Community

Filter your data

- > admin
- > config
- < humanResourceDB
 - employees
- > local

Collections

CREATE COLLECTION

Create Collection

Collection Name

Capped Collection
Fixed-size collections that support high-throughput operations that insert and retrieve documents based on insertion order. ⓘ

Use Custom Collation
Collation allows users to specify language-specific rules for string comparison, such as rules for lettercase and accent marks. ⓘ

Time-Series
Time-series collections efficiently store sequences of measurements over a period of time.

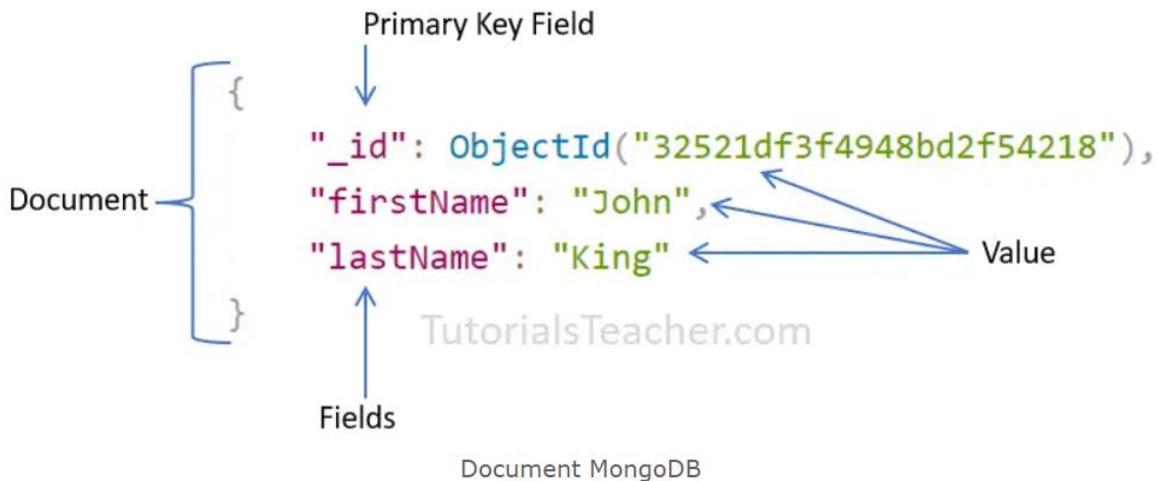
Create Collection

Boussole MongoDB - Collections

MongoDB : document, tableau, document intégré

Dans la base de données RDBMS, une table peut avoir plusieurs lignes et colonnes. De même dans MongoDB, une collection peut avoir plusieurs documents équivalents aux lignes. Chaque document a plusieurs "champs" qui sont équivalents aux colonnes. Donc, en termes simples, chaque document MongoDB est un enregistrement et une collection est une table qui peut stocker plusieurs documents.

Voici un exemple de document basé sur JSON.



Document MongoDB

Dans l'exemple ci-dessus, un document est contenu dans les accolades. Il contient plusieurs champs au "field":"value" format. Au-dessus, `_id`, `firstName` et `lastName` sont des noms de champs avec leurs valeurs respectives après deux-points `:`. Les champs sont séparés par une virgule. Une seule collection peut avoir plusieurs documents de ce type séparés par une virgule.

Le tableau suivant pour comprendre la relation entre la base de données, les collections et les documents.

Database	humanResourcedb Tutorialspoint.com		
Collections	employees	addresses	departments
Documents	<pre>["firstName": "John", "lastName": "King", "email": "john.king@abc.com", "salary": "33000"]</pre>	<pre>{ "street": "H12", "house": "33", "city": "New York", "country": "USA" }</pre>	<pre>{ "name": "Technical", "totalEmployees": "100" }</pre>

Base de données, collection et document MongoDB

Voici un exemple de document contenant un tableau et un document incorporé.

```
{
  "_id": ObjectId("32521df3f4948bd2f54218"),
  "firstName": "John",
  "lastName": "King",
  "email": "john.king@abc.com",
  "salary": "33000",
  "DoB": new Date('Mar 24, 2011'),
  "skills": [ "Angular", "React", "MongoDB" ],
  "address": {
    "street": "Upper Street",
    "house": "No 1",
    "city": "New York",
    "country": "USA"
  }
}
```

Le document MongoDB stocke les données au format JSON. Dans le document ci-dessus, "firstName", "lastName", "email" et "salary" sont les champs (comme les colonnes d'une table dans RDBMS) avec leurs valeurs correspondantes (par exemple, la valeur d'une colonne dans une ligne). Considérez `_id`field comme un champ de clé primaire qui stocke un `ObjectId` unique . `skills`est un tableau et `address`contient un autre document JSON.

Les noms de champs peuvent être spécifiés sans guillemets, comme illustré ci-dessous.

```
{
  _id: ObjectId("32521df3f4948bd2f54218"),
  firstName: "John",
  lastName: "King",
  email: "john.king@abc.com",
  salary: "33000",
  DoB: new Date('Mar 24, 2011'),
  skills: [ "Angular", "React", "MongoDB" ],
  address: {
    street:"Upper Street",
    house:"No 1",
    city:"New York",
    country:"USA"
  }
}
```

MongoDB stocke les données dans des paires clé-valeur en tant que document BSON. BSON est une représentation binaire d'un document JSON qui prend en charge plus de types de données que JSON. Les pilotes MongoDB convertissent le document JSON en données BSON.



Les points importants:

- MongoDB réserve `_idle` nom pour une utilisation en tant que champ de clé primaire unique contenant le type ObjectId. Cependant, vous êtes libre de donner n'importe quel nom avec n'importe quel type de données autre que le tableau.
- Un nom de champ de document ne peut pas être, `null` mais la valeur peut l'être.
- La plupart des documents MongoDB ne peuvent pas avoir de noms de champ en double. Cependant, cela dépend du pilote que vous utilisez pour stocker un document dans votre application.
- Les champs d'un document peuvent être sans guillemets " " s'ils ne contiennent pas d'espaces, par exemple `{ name: "Steve" }, { "first name": "Steve" }` sont des champs valides.
- Utilisez la notation par points pour accéder aux éléments du tableau ou aux documents incorporés.
- MongoDB prend en charge une taille de document maximale de 16 Mo. Utilisez [GridFS](#) pour stocker un document de plus de 16 Mo.
- Les champs d'un document BSON sont ordonnés. Cela signifie que l'ordre des champs est important lors de la comparaison de deux documents, par exemple `{x: 1, y: 2}` n'est pas égal à `{y: 2, x: 1}`
- MogoDB garde l'ordre des champs sauf `_id` qui est toujours le premier champ.
- La collection MongoDB peut stocker des documents avec différents champs. Il n'applique aucun schéma.

Documents intégrés :

Un document dans MongoDB peut avoir des champs qui contiennent un autre document. Il est également appelé documents imbriqués.

Ce qui suit est un document incorporé où le champ `department` et `address` contiennent un autre document.

```
{
  _id: ObjectId("32521df3f4948bd2f54218"),
  firstName: "John",
  lastName: "King",
  department: {
    _id: ObjectId("55214df3f4948bd2f8753"),
    name: "Finance"
  },
  address: {
    phone: { type: "Home", number: "111-000-000" }
  }
}
```

Dans le document intégré ci-dessus, notez que le `address` champ contient le `phone` champ qui contient un document de second niveau.

- Un document incorporé peut contenir jusqu'à 100 niveaux d'imbrication.
- Prend en charge une taille maximale de 16 Mo.
- Les documents intégrés peuvent être consultés en utilisant la notation par points `embedded-document.fieldname`, par exemple accéder au numéro de téléphone en utilisant `address.phone.number`.

Déployer

Un champ dans un document peut contenir un tableau. Les tableaux peuvent contenir tout type de données ou de documents intégrés.

Les éléments de tableau d'un document sont accessibles à l'aide de la notation par points avec la position d'index basée sur zéro et placés entre guillemets.

```
{
  _id: ObjectId("32521df3f4948bd2f54218"),
  firstName: "John",
  lastName: "King",
  email: "john.king@abc.com",
  skills: [ "Angular", "React", "MongoDB" ],
}
```

CHAPITRE 3 : MANIPULATION DES DONNEES

MongoDB - Insérer un seul document dans une collection à l'aide de insertOne()

Dans MongoDB, une collection représente une table dans RDBMS et un document est comme un enregistrement dans une table. Découvrez comment insérer un seul document dans une collection.

MongoDB fournit les méthodes suivantes pour insérer des documents dans une collection :

1. [insertOne\(\)](#) - Insère un seul document dans une collection.
2. [insert\(\)](#) - Insère un ou plusieurs documents dans une collection.
3. [insertMany\(\)](#) - Insère plusieurs documents dans une collection.

`insertOne()`

Utilisez la `db.<collection>.insertOne()` méthode pour insérer un seul document dans une collection. `db` pointe vers la base de données actuelle, `<collection>` existe ou porte un nouveau nom de collection.

Syntaxe:

`db.collection.insertOne(document, [writeConcern])`

Paramètres:

1. `document` : Un document à insérer dans la collection.
2. `writeConcern` : Facultatif. Un document exprimant le [problème d'écriture](#) pour remplacer le problème d'écriture par défaut.

Ce qui suit insère un document dans `employees` la collection.

```
db.employees.insertOne({
  firstName: "John",
  lastName: "King",
  email: "john.king@abc.com"
})
```

Sortir

```
{
  acknowledged: true,
  insertedId: ObjectId("616d44bea861820797edd9b0")
}
```

Dans l'exemple ci-dessus, nous avons passé un document à la `insertOne()` méthode. Notez que nous n'avons pas spécifié `_id` de champ. Ainsi, MongoDB insère un document dans une collection avec le champ unique généré automatiquement `_id`. Il renvoie un objet avec un champ booléen `acknowledged` qui indique si l'opération d'insertion a réussi ou non, et `insertedId` un champ avec la `_id` valeur nouvellement insérée.

Ce qui suit montre l'opération d'insertion dans le shell mongosh :



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
humanResourceDB> db.employees.insertOne({firstName:"John",lastName:"King",email:"john.king@abc.com"})
{
  acknowledged: true,
  insertedId: ObjectId("616d4628a861820797edd9b1")
}
humanResourceDB>
```

InsertOne() dans mongosh Shell

Utilisez le `find()` pour répertorier toutes les données d'une collection et la `pretty()` méthode pour formater les données résultantes.

`db.employees.find().pretty()`

Sortir

```
{
  _id: ObjectId("616d44bea861820797edd9b0"),
  firstName: "John",
  lastName: "King",
  email: "john.king@abc.com"
}
```

MongoDB est une base de données NoSQL. Ainsi, il n'applique pas de schéma à aucune collection. Cela signifie que vous pouvez insérer un document avec n'importe quel champ dans une collection. Par exemple, ce qui suit insère un document avec différents champs dans la `employees` collection.

```
db.employees.insertOne({
  fName: "John",
  lName: "King",
  emailid: "john.king@abc.com"
})
```

Sortir

```
{
  acknowledged: true,
  insertedId: ObjectId("546d44bea861820797ed214")
}
```

Il est recommandé de garder les mêmes noms de champs dans tous les documents d'une collection pour les gérer facilement.

Insérer _id manuellement

Il n'est pas nécessaire d'insérer une valeur générée automatiquement `_id`. Vous pouvez spécifier manuellement une valeur unique pour le `_id` champ, comme indiqué ci-dessous.

```
db.employees.insertOne({
  _id:"1",
  firstName: "John",
  lastName: "King",
  email: "john.king@abc.com"
})
{
  acknowledged: true,
  insertedId: 1
}
```

Notez que lors de l'ajout de votre valeur personnalisée au `_id` champ, une valeur doit être unique ; sinon, il lancera une erreur. Ce qui suit essaie d'ajouter la même `_id` valeur.

Exemple : Insérer un document

```
db.employees.insertOne({
  _id:"1",
  firstName: "John",
  lastName: "King",
  email: "john.king@abc.com"
})
```

Sortir

```
MongoServerError: E11000 duplicate key error collection: humanResourceDB.employees index: _id_ dup key: { _id: "1" }
```

insérer()

La `db.<collection>.insert()` méthode insère un document ou un tableau de documents dans une collection.

Syntaxe:

```
db.collection.insert(
```

```
    document or array of documents,  
    [writeConcern],  
    [ordered]
```

```
)
```

Paramètres:

1. document ou tableau de documents : document unique ou tableau de documents à insérer dans la collection.
2. writeConcern : Facultatif. Un document exprimant le [problème d'écriture](#) pour remplacer le problème d'écriture par défaut.
3. commandé : Facultatif. Une valeur booléenne indiquant s'il s'agit d'une opération d'insertion ordonnée ou non ordonnée. Si `true` ensuite effectue une insertion ordonnée où si une erreur se produit avec l'un des documents, MongoDB reviendra sans traiter les documents restants dans le tableau. Si `false`, traitez tous les documents même si une erreur s'est produite. La valeur par défaut est `true`.

Ce qui suit insère un seul document. C'est la même chose que la `insertOne()` méthode.

Exemple : insérer()

Copie

```
db.employees.insert({  
    firstName: "John",  
    lastName: "King",  
    email: "john.king@abc.com"  
})
```

Sortir

```
{
```

```

acknowledged: true,
insertedIds: { '0': ObjectId("616d62d9a861820797edd9b2") }
}

```

Ce qui suit insère plusieurs documents dans une collection en transmettant un tableau de documents.

Exemple : Insérer un document

Copie

```

db.employees.insert(
[
  {
    firstName: "John",
    lastName: "King",
    email: "john.king@abc.com"
  },
  {
    firstName: "Sachin",
    lastName: "T",
    email: "sachin.t@abc.com"
  },
  {
    firstName: "James",
    lastName: "Bond",
    email: "jamesb@abc.com"
  }
])

```

Sortir

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("616d63eda861820797edd9b3"),
    '1': ObjectId("616d63eda861820797edd9b4"),
    '2': ObjectId("616d63eda861820797edd9b5")
  }
}
```

Vous pouvez spécifier une `_id` valeur de champ différente dans un ou plusieurs documents, comme indiqué ci-dessous.

Exemple : Insérer un document

```
db.employees.insert([
  {
    firstName: "John",
    lastName: "King",
    email: "john.king@abc.com"
  },
  {
    _id:1,
    firstName: "Sachin",
    lastName: "T",
    email: "sachin.t@abc.com"
  },
  {
    firstName: "James",
    lastName: "Bond",
    email: "jamesb@abc.com"
  },
])

```

Sortir

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("616d63eda861820797edd9b3"),
    '1': 1,
    '2': ObjectId("616d63eda861820797edd9b5")
  }
}
```

Par défaut, la `insert()` méthode effectue des insertions ordonnées. Ainsi, si une erreur s'est produite dans l'un des documents, il ne traitera pas les documents restants. Par exemple, ce qui suit essaie d'insérer un deuxième document avec une `_id` valeur existante.

Exemple : Insérer un document

```
db.employees.insert([
  {
    firstName: "Steve",
    lastName: "J",
    email: "steve.j@abc.com"
  },
  {
    _id:1,
    firstName: "Kapil",
    lastName: "D",
    email: "kapil.d@abc.com"
  },
  {
    firstName: "Amitabh",
    lastName: "B",
    email: "amitabh.b@abc.com"
  },
])

```

Sortir

Uncought:

```
MongoBulkWriteError: E11000 duplicate key error collection: humanResourceDB.employees
index: _id_ dup key: { _id: 1 }
Result: BulkWriteResult {
  result: {
    ok: 1,
    writeErrors: [
      WriteError {
        err: {
          index: 1,
          code: 11000,
          errmsg: 'E11000 duplicate key error collection: humanResourceDB.employees index: _id_
dup key: { _id: 1 }',
          errInfo: undefined,
          op: {
            _id: 1,
            firstName: 'Kapil',
            lastName: 'D',

```

```

        email: 'kapil.d@abc.com'
    }
}
]
],
writeConcernErrors: [],
insertedIds: [
    { index: 0, _id: ObjectId("616e6b7e3fa8bd4420d49371") },
    { index: 1, _id: 1 },
    { index: 2, _id: ObjectId("616e6b7e3fa8bd4420d49372") }
],
nInserted: 1,
nUpserted: 0,
nMatched: 0,
nModified: 0,
nRemoved: 0,
upserted: []
}
}
}
```

Dans la sortie ci-dessus, il rencontre une erreur lors de l'insertion du deuxième document. Ainsi, il insère uniquement le premier document. **nInserted** indique le nombre de documents insérés avec succès.

Spécifiez l'`{ordered:false}` de l'insertion pour déclassement à effectuer qui insérera tous les documents valides même en cas d'erreur.

Exemple : Insérer un document

Copie

```
db.employees.insert(
[
    {
        firstName: "Steve",
        lastName: "J",
        email: "steve.j@abc.com"
    },
    {
        _id:1,
        firstName: "Kapil",
        lastName: "D",
    }
]
```

```

    email: "kapil.d@abc.com"
},
{
  firstName: "Amitabh",
  lastName: "B",
  email: "amitabh.b@abc.com"
},
],
{ ordered: false}
)
Sortir

```

Uncought:

```
MongoBulkWriteError: E11000 duplicate key error collection: humanResourceDB.employees
index: _id_ dup key: { _id: 1 }
```

```
Result: BulkWriteResult {
```

```

  result: {
    ok: 1,
    writeErrors: [
      WriteError {
        err: {
          index: 1,
          code: 11000,
          errmsg: 'E11000 duplicate key error collection: humanResourceDB.employees index: _id_
dup key: { _id: 1 }',
          errInfo: undefined,
          op: {
            _id: 1,
            firstName: 'Kapil',
            lastName: 'D',
            email: 'kapil.d@abc.com'
          }
        }
      }
    ],
    writeConcernErrors: [],
    insertedIds: [
      { index: 0, _id: ObjectId("616e6be33fa8bd4420d49373") },
      { index: 1, _id: 1 },
      { index: 2, _id: ObjectId("616e6be33fa8bd4420d49374") }
    ],
  }
}
```

```

nInserted: 2,
nUpserted: 0,
nMatched: 0,
nModified: 0,
nRemoved: 0,
upserted: []
}
}

```

insertMany()

Le `db.<collection>.insertMany()` insère plusieurs documents dans une collection. Il ne peut pas insérer un seul document.

Syntaxe:

```

db.collection.insertMany(
  [document1, document2, ....],
  {
    writeConcern: <document>,
    ordered: <boolean>
  })

```

Ce qui suit ajoute plusieurs documents à l'aide de la `insertMany()` méthode.

Exemple : Insérer un document

```

db.employees.insertMany(
  [
    {
      firstName: "John",
      lastName: "King",
      email: "john.king@abc.com"
    },
    {
      firstName: "Sachin",
      lastName: "T",
      email: "sachin.t@abc.com"
    },
    {
      firstName: "James",
      lastName: "Bond",
    }
  ]
)

```

```

    email: "jamesb@abc.com"
  },
])
Sortir
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("616d63eda861820797edd9b3"),
    '1': ObjectId("616d63eda861820797edd9b4"),
    '2': ObjectId("616d63eda861820797edd9b5")
  }
}

```

Ce qui suit insère `_id` des valeurs personnalisées.

Exemple : insertMany() avec Custom `_id`

```

db.employees.insertMany([
  {
    _id:1,
    firstName: "John",
    lastName: "King",
    email: "john.king@abc.com",
    salary: 5000
  },
  {
    _id:2,
    firstName: "Sachin",
    lastName: "T",
    email: "sachin.t@abc.com",
    salary: 8000
  },
  {
    _id:3,
    firstName: "James",
    lastName: "Bond",
    email: "jamesb@abc.com",
    salary: 7500
  },
  {
    _id:4,

```

```

firstName: "Steve",
lastName: "J",
email: "steve.j@abc.com",
salary: 9000

},
{
  _id:5,
  firstName: "Kapil",
  lastName: "D",
  email: "kapil.d@abc.com",
  salary: 4500

},
{
  _id:6,
  firstName: "Amitabh",
  lastName: "B",
  email: "amitabh.b@abc.com",
  salary: 11000
}
])
Sortir

{
  acknowledged: true,
  insertedIds: {'0': 1, '1': 2, '2': 3, '3': 4, '4': 5, '5': 6 }
}

```

Utilisez le `ordered` paramètre de la même manière que la `insert()` méthode pour traiter tous les documents même si une erreur s'est produite.

Remarque : Si la collection utilisée avec ces méthodes n'existe pas, elles créent la collection spécifiée. MongoDB est sensible à la casse, `employees` et `Employee` sont donc considérés comme deux collections différentes.

MongoDB : rechercher un seul document dans une collection à l'aide de findOne()

Dans MongoDB, une collection représente une table dans RDBMS et un document est comme un enregistrement dans une table. Ici, vous apprendrez comment récupérer ou trouver un seul document d'une collection.

MongoDB propose deux méthodes pour rechercher des documents dans une collection :

1. [findOne\(\)](#) - renvoie le premier document correspondant aux critères spécifiés.
2. [find\(\)](#) - renvoie un curseur vers les documents sélectionnés qui correspondent aux critères spécifiés.

Ce qui suit insère des documents dans la `employees` collection.

Exemple de données

Copie

```
db.employees.insertMany([
  {
    _id:1,
    firstName: "John",
    lastName: "King",
    email: "john.king@abc.com",
    salary: 5000
  },
  {
    _id:2,
    firstName: "Sachin",
    lastName: "T",
    email: "sachin.t@abc.com",
    salary: 8000
  },
  {
    _id:3,
    firstName: "James",
    lastName: "Bond",
    email: "jamesb@abc.com",
    salary: 7500
  },
  {
    _id:4,
```

```

firstName: "Steve",
lastName: "J",
email: "steve.j@abc.com",
salary: 9000

},
{
  _id:5,
  firstName: "Kapil",
  lastName: "D",
  email: "kapil.d@abc.com",
  salary: 4500

},
{
  _id:6,
  firstName: "Amitabh",
  lastName: "B",
  email: "amitabh.b@abc.com",
  salary: 11000
}
])
)

```

trouverUn()

La `findOne()` méthode renvoie le premier document correspondant aux critères spécifiés.

Syntaxe:

`db.collection.findOne(query, projection)`

Paramètres:

1. requête : Facultatif. Spécifie les critères à l'aide d'opérateurs de requête.
2. saillie : Facultatif. Spécifie les champs à inclure dans un document résultant.

Le `findOne()` renvoie le premier document d'une collection si aucun paramètre n'est passé.

Exemple : trouver()

Copie

`db.employees.findOne()`

Sortir

```
{
  _id: 1,
  firstName: 'John',
  lastName: 'King',
  email: 'john.king@abc.com',
  salary: 5000
}
```

Spécifiez un critère { field: "value", field:"value",..} sur lequel vous souhaitez rechercher un document. Par exemple, ce qui suit renvoie un document où `firstName` champ est "Kapil".

Exemple : trouver()

Copie

```
db.employees.findOne({firstName: "Kapil"})
```

Sortir

```
{
  _id: 5,
  firstName: 'Kapil',
  lastName: 'D',
  email: 'kapil.d@abc.com',
  salary: 4500
}
```

La `findOne()` méthode effectue la recherche sensible à la casse, donc `{firstName: "Kapil"}` et `{firstName: "kapil"}` renvoie un résultat différent.

Il renvoie null s'il ne trouve pas de document correspondant aux critères spécifiés.

Exemple : trouver()

Copie

```
db.employees.findOne({_id:10})
```

Sortir

null

Utilisez les opérateurs de requête pour une recherche plus précise. Par exemple, ce qui suit trouve le premier document où le salaire est supérieur à 8000.

Exemple : trouver()

Copie

```
db.employees.findOne({ salary: {$gt: 8000}})
```

Sortir

```
{
  _id: 4,
  firstName: 'Steve',
  lastName: 'J',
  email: 'steve.j@abc.com',
  salary: 9000
}
```

Dans l'exemple ci-dessus, les critères de l'opérateur de requête pour le `salary` champ sont écrits dans un autre document sous la forme `{field: {operator: value}}`. Le `{salary: {$gt: 8000}}` critère renvoie le premier document où `salary` est supérieur à 8000.

Projection

Utilisez le paramètre de projection pour spécifier les champs à inclure dans le résultat. Le format du paramètre de projection est `{<field>: <1 or true>, <field>: <1 or true>...}` où 1 ou vrai inclut le champ, et 0 ou faux exclut le champ dans le résultat.

Exemple : trouver()

Copie

```
db.employees.findOne({firstName: "Sachin"}, {firstName:1, lastName:1})
```

Sortir

```
{ _id: 2, firstName: 'Sachin', lastName: 'T' }
```

Notez que par défaut, le `_id` champ sera inclus dans le résultat. Pour l'omettre, spécifiez `{ _id:0 }` dans la projection.

Exemple : trouver()

Copie

```
db.employees.findOne({firstName: "Sachin"}, {_id: 0, firstName:1, lastName:1})
```

Sortir

```
{firstName: 'Sachin', lastName: 'T' }
```

MongoDB : rechercher des documents dans la collection à l'aide de find()

Dans MongoDB, une collection représente une table dans RDBMS et un document est comme un enregistrement dans une table. Vous apprendrez ici comment récupérer ou retrouver un ou plusieurs documents d'une collection.

MongoDB propose deux méthodes pour rechercher des documents dans une collection :

1. [findOne\(\)](#) - renvoie le premier document correspondant aux critères spécifiés.
2. [find\(\)](#) - renvoie un curseur vers les documents sélectionnés qui correspondent aux critères spécifiés.

Ce qui suit insère des documents dans la `employees` collection.

Exemple de données

Copie

```
db.employees.insertMany([
  {
    _id:1,
    firstName: "John",
    lastName: "King",
    email: "john.king@abc.com",
    salary: 5000,
    skills: [ "Angular", "React", "MongoDB" ],
    department: {
      "name": "IT"
    }
  },
  {
    _id:2,
    firstName: "Sachin",
    lastName: "T",
    email: "sachin.t@abc.com",
    salary: 8000,
    skills: [ "Accounting", "Tax" ],
    department: {
      "name": "Finance"
    }
  }
])
```

```

},
{
  _id:3,
  firstName: "James",
  lastName: "Bond",
  email: "jamesb@abc.com",
  salary: 7500,
  skills: [ "Sales", "Marketing" ],
  department: {
    "name": "Marketing"
  }
},
{
  _id:4,
  firstName: "Steve",
  lastName: "J",
  email: "steve.j@abc.com",
  salary: 7000

},
{
  _id:5,
  firstName: "Kapil",
  lastName: "D",
  email: "kapil.d@abc.com",
  salary: 4500,
  skills: [ "Accounting", "Tax" ],
  department: {
    "name": "Finance"
  }
},
{
  _id:6,
  firstName: "Amitabh",
  lastName: "B",
  email: "amitabh.b@abc.com",
  salary: 7000
}
])

```

Rechercher des documents à l'aide de la méthode find()

La `find()` méthode trouve tous les documents qui correspondent aux critères spécifiés et renvoie l'objet curseur. L'objet curseur est un pointeur vers le jeu de résultats.

Syntaxe:

```
db.collection.find(query, projection)
```

Paramètres:

1. requête : Facultatif. Spécifie les critères à l'aide d'opérateurs de requête.
2. saillie : Facultatif. Spécifie les champs à inclure dans un document résultant.

Le `find()` renvoie tous les documents d'une collection si aucun paramètre n'est passé. Ce qui suit exécute le `db.employees.find()` shell in mongosh :

```
humanResourceDB> db.employees.find()
[
  {
    _id:1,
    prénom : "John",
    nom : "Roi",
    e-mail : " john.king@abc.com ",
    salaire : 5000,
    compétences : [ "Angular", "React", "MongoDB" ],
    service : { "name":"IT" }
  },
  {
    _id:2,
    prénom : "Sachin",
    nom: "T",
    email : " sachin.t@abc.com ",
    salaire : 8000,
  }
]
```

```
compétences : [ "Comptabilité", "Fiscalité" ],  
département : { "name":"Finances" }  
,  
{  
_id:3,  
prénom : "James",  
nom de famille : "Bond",  
courriel : " jamesb@abc.com ",  
salaire : 7500,  
compétences : [ "Vente", "Marketing" ],  
service : { "name":"Marketing" }  
,  
{  
_id:4,  
prénom : "Steve",  
nom: "J",  
e-mail : " steve.j@abc.com ",  
salaire : 7000  
  
,  
{  
_id:5,  
prénom : "Kapil",  
nom: "D",  
email : " kapil.d@abc.com ",  
salaire : 4500,  
compétences : [ "Comptabilité", "Fiscalité" ],  
département : { "name":"Finances" }
```

```

},
{
  _id:6,
  prénom : "Amitabh",
  nom: "B",
  email : " amitabh.b@abc.com ",
  salaire : 7000
}
]
```

L'exécution `db.collection.find()` dans **mongosh shell** itère automatiquement le curseur pour afficher jusqu'aux 20 premiers documents. Tapez `it` pour continuer l'itération.

Vous pouvez spécifier les critères selon `{ field: "value", field:"value",.. }` lesquels vous souhaitez rechercher un document. Par exemple, ce qui suit renvoie tous les documents où `salary` est 7000.

Exemple : trouver(critère)

Copie

```
db.employees.find({ salary: 7000 })
```

```
[{
```

```
  _id:4,
  firstName: "Steve",
  lastName: "J",
  email: "steve.j@abc.com",
  salary: 7000
```

```
},
```

```
{
```

```
  _id:6,
  firstName: "Amitabh",
  lastName: "B",
  email: "amitabh.b@abc.com",
```

```

    salary: 7000
}
]
```

La `find()` méthode effectue la recherche sensible à la casse, donc `{firstName: "Kapil"}` et `{firstName: "kapil"}` renvoie un résultat différent.

Utilisez les opérateurs de requête pour une recherche plus précise. Par exemple, ce qui suit trouve le premier document où le salaire est supérieur à 8000.

Exemple : `find()` avec l'opérateur de requête

Copie

```
db.employees.find({ salary: {$gt: 7000} })
```

Sortir

```
[{
  _id: 2,
  firstName: 'Sachin',
  lastName: 'T',
  email: 'sachin.t@abc.com',
  salary: 8000,
  skills: [ 'Accounting', 'Tax' ],
  department: { name: 'Finance' }
},
{
  _id: 3,
  firstName: 'James',
  lastName: 'Bond',
  email: 'jamesb@abc.com',
  salary: 7500,
  skills: [ 'Sales', 'Marketing' ],
  department: { name: 'Marketing' }
}]
```

Spécifiez plusieurs critères en ajoutant un autre opérateur de requête en tant que champ. Ce qui suit récupère les documents dont `salary` le champ est supérieur à 7000 et inférieur à 8000.

Exemple : `find()` avec opérateur de requête multiple

Copie

```
db.employees.find({ salary: { $gt: 7000, $lt: 8000 } })
```

Sortir

```
[{
```

```

_id: 3,
firstName: 'James',
lastName: 'Bond',
email: 'jamesb@abc.com',
salary: 7500,
skills: [ 'Sales', 'Marketing' ],
department: { name: 'Marketing' }
}]

```

Interroger le document intégré

Vous pouvez spécifier des critères pour les champs de document intégrés à l'aide de la notation par points, comme indiqué ci-dessous.

Exemple : find() avec opérateur de requête multiple

Copie

```
db.employees.find({ "department.name": "Finance" })
```

Sortir

```
[
{
  _id: 2,
  firstName: 'Sachin',
  lastName: 'T',
  email: 'sachin.t@abc.com',
  salary: 8000,
  skills: [ 'Accounting', 'Tax' ],
  department: { name: 'Finance' }
},
{
  _id: 5,
  firstName: 'Kapil',
  lastName: 'D',
  email: 'kapil.d@abc.com',
  salary: 4500,
  skills: [ 'Accounting', 'Tax' ],
  department: { name: 'Finance' }
}]
```

Interroger les éléments du tableau

Vous pouvez rechercher des documents en fonction de l'élément de tableau, de l'index ou de la taille.

Exemple : find() sur Array

Copie

```
db.employees.find({ "skills": "Tax" }) //returns documents where skills contains "Tax"
db.employees.find({ "skills": { $in: [ "Tax", "Sales" ] } }) //returns documents where skills
contains "Tax" or "Sales"
db.employees.find({ "skills": { $all: [ "Tax", "Accounting" ] } }) //returns documents where skills
contains "Tax" and "Accounting"
db.employees.find({ "skills": { $size: 3 } }) //returns documents where skills contains 3 elements
```

Projection

Utilisez le paramètre de projection pour spécifier les champs à inclure dans le résultat. Le format du paramètre de projection est {<field>: <1 or true>, <field>: <1 or true>...} où 1 ou vrai inclut le champ, et 0 ou faux exclut le champ dans le résultat.

Exemple : trouver()

Copie

```
db.employees.find({ salary: 7000 }, { firstName:1, lastName:1 })
```

Sortir

```
[{
  _id:4,
  firstName: "Steve",
  lastName: "J",
},
{
  _id:6,
  firstName: "Amitabh",
  lastName: "B",
}]
```

Notez que par défaut, le `_id` champ sera inclus dans le résultat. Pour l'omettre, spécifiez `{ _id:0 }` dans la projection.

Exemple : trouver()

Copie

```
db.employees.findOne({firstName: "Sachin"}, {_id: 0, firstName:1, lastName:1})
```

Sortir

```
[{
  firstName: "Steve",
  lastName: "J",
},
{
  firstName: "Amitabh",
  lastName: "B",
}]
```

Curseur MongoDB

La `find()` méthode renvoie un objet curseur qui peut être utilisé pour itérer le résultat.

L'exemple suivant récupère l'objet curseur et l'affecte à une variable.

Exemple : objet curseur

Copie

```
var cursor = db.employees.find()
```

L'objet curseur a les méthodes importantes suivantes :

Méthode	Description
curseur.count()	Renvoie le nombre total de documents référencés par un curseur.
curseur.forEach()	Itère le curseur pour appliquer une fonction JavaScript à chaque document à partir du curseur.
curseur.hasNext()	Renvoie true si un curseur peut itérer davantage pour renvoyer plus de documents.
curseur.isExhausted()	Renvoie vrai si le curseur est fermé et qu'il ne reste aucun objet dans le lot.
curseur.itcount()	Compte le nombre de documents restant dans un curseur.
curseur.limit()	Spécifiez le nombre maximum de documents que le curseur renverra.

Méthode	Description
curseur.map()	Applique une fonction à chaque document visité par le curseur et collecte les valeurs de retour des applications successives de la fonction dans un objet Cursor.
curseur.max()	Spécifie la limite supérieure exclusive pour un index spécifique afin de contraindre les résultats de find().
curseur.min()	Spécifie la limite inférieure inclusive pour un index spécifique afin de contraindre les résultats de find().
curseur.suivant()	Renvoie le document suivant à partir du jeu de résultats.
curseur.pretty()	Afficher le résultat dans un format lisible.
curseur.readConcern()	Spécifie un niveau d'isolement pour les opérations de lecture.
curseur.skip()	Ignore le nombre spécifié de documents pour la pagination.
curseur.sort()	Spécifie l'ordre dans lequel la requête renvoie les documents correspondants.
curseur.toArray()	Renvoie un tableau qui contient tous les documents d'un curseur.

Notez que les méthodes de curseur dépendent des pilotes que vous utilisez dans votre application. Visitez **les méthodes de curseur mongosh** pour plus d'informations.

L'exemple suivant montre comment utiliser `next()` la méthode dans le shell mongosh.

```
humanResourceDB> var cur = db.employees.find()
```

```
humanResourceDB> cur.next()
```

```
{
  _id:1,
  prénom : "John",
  nom : "Roi",
  e-mail : " john.king@abc.com ",
  salaire : 5000
}
```

```
humanResourceDB> cur.next()
```

```
{
  _id:2,
```

```
prénom : "Sachin",
nom: "T",
email : " sachin.t@abc.com ",
salaire : 8000

}

humanResourceDB> cur.next()

{
    _id:3,
    prénom : "James",
    nom de famille : "Bond",
    courriel : " jamesb@abc.com ",
    salaire : 7500

}

humanResourceDB> cur.next()

{
    _id:4,
    prénom : "Steve",
    nom: "J",
    e-mail : " steve.j@abc.com ",
    salaire : 7000

}

humanResourceDB> cur.next()

{
    _id:5,
    prénom : "Kapil",
    nom: "D",
    email : " kapil.d@abc.com ",
```

```

salaire : 4500

}

humanResourceDB> cur.next()

{
  _id:6,
  prénom : "Amitabh",
  nom: "B",
  email : " amitabh.b@abc.com ",
  salaire : 7000
}

humanResourceDB> cur.next()

nul

humanResourceDB> cur.next()

MongoCursorExhaustedError : le curseur est épuisé

```

Dans l'exemple ci-dessus, si le curseur atteint la fin, il sera `MongoCursorExhaustedError: Cursor is exhausted.` Utilisez la `hasNext()` méthode avant d'appeler le `next()` pour éviter une erreur.

Utilisez `cursor.forEach()` la méthode pour itérer le résultat. Spécifiez la méthode intégrée `printjson` pour imprimer le résultat, comme indiqué ci-dessous.

```
humanResourceDB> var cur = db.employees.find()
```

```

humanResourceDB> cur.forEach(printjson)

{
  _id:1,
  prénom : "John",
  nom : "Roi",
  e-mail : " john.king@abc.com ",
```

```
salaire : 5000
}
{
_id:2,
prénom : "Sachin",
nom: "T",
email : " sachin.t@abc.com ",
salaire : 8000

}

{
_id:3,
prénom : "James",
nom de famille : "Bond",
courriel : " jamesb@abc.com ",
salaire : 7500

}

{
_id:4,
prénom : "Steve",
nom: "J",
e-mail : " steve.j@abc.com ",
salaire : 7000

}

{
_id:5,
prénom : "Kapil",
nom: "D",
```

```
email : " kapil.d@abc.com ",  
salaire : 4500  
  
}  
{  
_id:6,  
prénom : "Amitabh",  
nom: "B",  
email : " amitabh.b@abc.com ",  
salaire : 7000  
}
```

Apprenez à trier des documents en utilisant `cursor.sort()` la méthode dans le chapitre suivant.

Trier les documents dans la collection MongoDB

MongoDB fournit la `db.collection.find()` méthode qui renvoie un objet curseur pour les documents résultants. Utilisez la `cursor.sort()` méthode ou `db.collection.find().sort()` pour trier les documents résultants dans un curseur en fonction de l'ordre spécifié.

Syntaxe:

```
db.collection.find().sort(document)
```

Paramètres:

- document : un document qui définit l'ordre de tri au `{field: 1, field:-1,...}` format où 1 correspond à l'ordre croissant et -1 à l'ordre décroissant.

Ce qui suit insère des documents dans la `employees` collection.

Exemple de données

Copie

```
db.employees.insertMany([
  {
    _id:1,
    firstName: "John",
    lastName: "King",
    email: "john.king@abc.com",
    salary: 5000,
    skills: [ "Angular", "React", "MongoDB" ],
    department: {
      "name": "IT"
    }
  },
  {
    _id:2,
    firstName: "Sachin",
    lastName: "T",
    email: "sachin.t@abc.com",
    salary: 8000,
    skills: [ "Accounting", "Tax" ],
    department: {
      "name": "Finance"
    }
  }
])
```

```

},
{
  _id:3,
  firstName: "James",
  lastName: "Bond",
  email: "jamesb@abc.com",
  salary: 7500,
  skills: [ "Sales", "Marketing" ],
  department: {
    "name": "Marketing"
  }
},
{
  _id:4,
  firstName: "Steve",
  lastName: "J",
  email: "steve.j@abc.com",
  salary: 7000

},
{
  _id:5,
  firstName: "Kapil",
  lastName: "D",
  email: "kapil.d@abc.com",
  salary: 4500,
  skills: [ "Accounting", "Tax" ],
  department: {
    "name": "Finance"
  }
},
{
  _id:6,
  firstName: "Amitabh",
  lastName: "B",
  email: "amitabh.b@abc.com",
  salary: 7000
}
])

```

L'exemple suivant trie la `employees` collection dans l'ordre croissant du `firstName` champ.

Exemple : trier()

Copie

```
db.employees.find().sort({ firstName:1 })
```

Sortir

```
[  
  {  
    _id: 6,  
    firstName: 'Amitabh',  
    lastName: 'B',  
    email: 'amitabh.b@abc.com',  
    salary: 7000  
  },  
  {  
    _id: 3,  
    firstName: 'James',  
    lastName: 'Bond',  
    email: 'jamesb@abc.com',  
    salary: 7500,  
    skills: [ 'Sales', 'Marketing' ],  
    department: { name: 'Marketing' }  
  },  
  {  
    _id: 1,  
    firstName: 'John',  
    lastName: 'King',  
    email: 'john.king@abc.com',  
    salary: 5000,  
    skills: [ 'Angular', 'React', 'MongoDB' ],  
    department: { name: 'IT' }  
  },  
  {  
    _id: 5,  
    firstName: 'Kapil',  
    lastName: 'D',  
    email: 'kapil.d@abc.com',  
    salary: 4500,  
    skills: [ 'Accounting', 'Tax' ],  
  }]
```

```

department: { name: 'Finance' }
},
{
  _id: 2,
  firstName: 'Sachin',
  lastName: 'T',
  email: 'sachin.t@abc.com',
  salary: 8000,
  skills: [ 'Accounting', 'Tax' ],
  department: { name: 'Finance' }
},
{
  _id: 4,
  firstName: 'Steve',
  lastName: 'J',
  email: 'steve.j@abc.com',
  salary: 7000
}
]

```

Vous pouvez utiliser l'objet curseur pour trier le résultat. Ce qui suit renvoie le même résultat que ci-dessus.

Exemple : curseur.sort()

Copie

```

var cursor = db.employees.find()
cursor.sort({ firstName:1 })

```

L'exemple suivant liste le tri sur différents champs.

Exemple : trier()

Copie

```

db.employees.find().sort({ _id: -1 })// sorts by descending order of _id
db.employees.find().sort({ salary: -1 })// sorts by descending order of salary
db.employees.find({salary: {$gt:5000}}).sort({ salary: -1 })// find where salary > 5000 and sorts
the result by descending order of salary
db.employees.find().sort({ firstName:1, salary: -1 })// sorts by ascending order of firstName and
descending order of salary
db.employees.find().sort({"department.name":1}) // sorts by embedded doc department.name

```

MongoDB utilise l'ordre de comparaison suivant, du plus bas au plus élevé pour comparer les valeurs de différents types de BSON.

1. MinKey (type interne)
2. Nul
3. Nombres (entiers, longs, doubles, décimaux)
4. Symbole, Chaîne
5. Objet
6. Déployer
7. BinData
8. ID d'objet
9. booléen
10. Date
11. Horodatage
12. Expression régulière
13. MaxKey (type interne)

Mettre à jour un seul document à l'aide de updateOne() dans MongoDB

Apprenez à mettre à jour un seul document en utilisant la `updateOne()` méthode dans MongoDB.

MongoDB fournit les méthodes suivantes pour mettre à jour les documents existants dans une collection :

- [`db.collection.updateOne\(\)`](#) - Modifie un seul document dans une collection.
- [`db.collection.updateMany\(\)`](#) - Modifie un ou plusieurs documents dans une collection.

`db.collection.updateOne()`

Utilisez la `db.<collection>.updateOne()` méthode pour mettre à jour un seul document dans une collection qui correspond aux critères de filtre spécifiés. Il met à jour le premier document correspondant même si plusieurs documents correspondent aux critères.

Syntaxe:

`db.collection.updateOne(filter, document, options)`

Paramètres:

1. filtre : Les critères de sélection pour la mise à jour, identiques à la méthode [`find\(\)`](#) .
2. document : document ou pipeline contenant des modifications à appliquer.

3. Options : Facultatif. Peut contenir des options pour le comportement de mise à jour. Cela inclut upsert, writeConcern, classement, etc.

Dans la syntaxe ci-dessus, dbpointe vers la base de données actuelle, <collection>points est un nom de collection existant.

Pour illustrer l'opération de mise à jour, insérez les exemples de documents suivants dans la employeescollection.

Exemple de données

Copie

```
db.employees.insertMany([
  {
    _id:1,
    firstName: "John",
    lastName: "King",
    email: "john.king@abc.com",
    salary: 5000
  },
  {
    _id:2,
    firstName: "Sachin",
    lastName: "T",
    email: "sachin.t@abc.com",
    salary: 8000
  },
  {
    _id:3,
    firstName: "James",
    lastName: "Bond",
    email: "jamesb@abc.com",
    salary: 7500
  },
  {
    _id:4,
    firstName: "Steve",
    lastName: "J",
```

```
email: "steve.j@abc.com",
salary: 7000
},
{
  _id:5,
  firstName: "Kapil",
  lastName: "D",
  email: "kapil.d@abc.com",
  salary: 4500
},
{
  _id:6,
  firstName: "Amitabh",
  lastName: "B",
  email: "amitabh.b@abc.com"
  salary: 7000
}
])

```

Mettre à jour un seul champ

Ce qui suit met à jour un seul champ dans un seul document de employeesla collection.

Exemple : updateOne()

Copie

```
db.employees.updateOne({ _id:1 }, { $set: { firstName:'Morgan' } })
```

Sortir

{

```
    acknowledged: true,  
    insertedId: null,  
    matchedCount: 1,  
    modifiedCount: 1,  
    upsertedCount: 0  
}
```

Dans l'exemple ci-dessus, le premier paramètre est le critère de filtre spécifié en tant que document, `{_id:1}` indique que trouver un document dont `_id` est 1. Le deuxième paramètre est utilisé pour spécifier les champs et les valeurs à modifier sur le document correspondant dans le `{<update-operator>: { field: value, field:value,... }}` format. Utilisez l'opérateur de mise à jour pour spécifier l'action à effectuer. Ici, nous voulons définir la valeur d'un champ, utilisez donc `$set` l'opérateur pour spécifier les champs et les valeurs mises à jour au `{field:updated}`

`value}`format. `{ $set: {firstName:'Morgan'}}`remplace le `firstName`par "Morgan"le premier document qui correspond aux critères spécifiés `{_id:1}`.

Dans la sortie, `matchedCount`indique le nombre de documents correspondant aux critères et `modifiedCount`indique le nombre de documents mis à jour. La `updateOne()`méthode modifiera toujours un seul document.

Maintenant, vérifiez s'il a mis à jour une valeur ou non en utilisant la `findOne()`méthode indiquée ci-dessous.

Vérifier le document mis à jour

Copie

```
db.employees.find({_id:1})
```

Sortir

```
{
  _id: 1,
  firstName: 'Morgan',
  lastName: 'King',
  email: 'john.king@abc.com',
  salary: 5000
}
```

La `updateOne()`méthode ajoute le champ spécifié s'il n'existe pas dans un document correspondant. Par exemple, ce qui suit ajoutera le `location`champ.

Exemple : `updateOne()`

Copie

```
db.employees.updateOne({firstName:"Steve"}, { $set: {location: "USA" }})
```

Sortir

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Exécutez la méthode suivante `findOne()`pour voir les données mises à jour.

Vérifier le document mis à jour

Copie

```
db.employees.find({firstName:"Steve"})
```

Sortir

```
{
  _id:4,
  firstName: "Steve",
  lastName: "J",
  email: "steve.j@abc.com",
  salary: 7000,
  location:"USA"
}
```

Utilisez l' `$inc` opérateur de mise à jour pour augmenter la valeur du champ du montant spécifié.

Exemple : Opérateur `$inc`

Copie

```
db.employees.updateOne({firstName:"Steve"}, { $inc: {salary: 500}})
```

Sortir

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Exécutez la méthode suivante `find()`pour voir les données mises à jour.

Vérifier le document mis à jour

Copie

```
db.employees.find({firstName:"Steve"})
```

Sortir

```
{
  _id:4,
  firstName: "Steve",
  lastName: "J",
  email: "steve.j@abc.com",
  salary: 7500,
  location:"USA"
}
```

Mettre à jour plusieurs champs

Vous pouvez également spécifier plusieurs champs à mettre à jour. `email`Les mises à jour et les champs suivants `lastName`.

Exemple : Mettre à jour plusieurs champs

Copie

```
db.employees.updateOne({_id:2}, { $set: {lastName:"Tendulkar",
email:"sachin.tendulkar@abc.com" }})
```

Sortir

```
{
acknowledged: true,
insertedId: null,
matchedCount: 1,
modifiedCount: 1,
upsertedCount: 0
}
```

Exécutez la méthode suivante `find()`pour voir les données mises à jour.

Vérifier le document mis à jour

Copie

```
db.employees.find({_id:2})
```

Sortir

```
{
_id:2,
firstName: "Sachin",
lastName: "Tendulkar",
email: "sachin.tendulkar@abc.com",
salary: 8000
}
```

La `updateOne()`méthode ne met à jour qu'un seul document, même si elle trouve plusieurs documents. Par exemple, ce qui suit met à jour le premier document même s'il renvoie plusieurs documents.

Exemple : `updateOne()`

Copie

```
db.employees.updateOne({salary:7000}, { $set: {salary:7500}})
```

Sortir

```
{
acknowledged: true,
insertedId: null,
```

```

matchedCount: 1,
modifiedCount: 1,
upsertedCount: 0
}

```

Dans l'exemple ci-dessus, la `employees` collection contient deux documents qui ont `salary:7000` un champ. Cependant, le `updateOne()` document modifié est le premier document à partir du résultat correspondant.

Upsert - Ajouter s'il n'existe pas

Spécifiez `{upsert:true}` comme troisième paramètre dans la `UpdateOne()` méthode. Le `upsert:true` ajoute un nouveau document si le document correspondant est introuvable.

Exemple : Upsert

Copie

```
db.employees.updateOne({firstName:"Heer"}, { $set: {lastName:"Patel", salary:2000}})
```

Sortir

```
{
  acknowledged: true,
  insertedId: ObjectId("6172a2e9ce7d5ca09d6ab082"),
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1
}
```

Dans l'exemple ci-dessus, MongoDB ajoute un nouveau document avec `new _id`, car il ne trouve pas de document avec le `firstName:"Heer"`.

Opérateurs de mise à jour

Le tableau suivant répertorie les opérateurs de mise à jour qui peuvent être utilisés avec les méthodes `updateOne()` et `updateMany()`.

Méthode	Description
<code>\$currentDate</code>	Définit la valeur d'un champ à la date actuelle, soit en tant que date, soit en tant qu'horodatage.
<code>\$inc</code>	Incrémente la valeur du champ du montant spécifié.
<code>\$min</code>	Ne met à jour le champ que si la valeur spécifiée est inférieure à la valeur du champ existant.

Méthode	Description
\$max	Ne met à jour le champ que si la valeur spécifiée est supérieure à la valeur du champ existant.
\$mul	Multiplie la valeur du champ par le montant spécifié.
\$ renommer	Renomme un champ.
\$ ensemble	Définit la valeur d'un champ dans un document.
\$setOnInsert	Définit la valeur d'un champ si une mise à jour entraîne l'insertion d'un document. N'a aucun effet sur les opérations de mise à jour qui modifient les documents existants.
\$unset	Supprime le champ spécifié d'un document.

Mettre à jour plusieurs documents à l'aide de updateMany() dans MongoDB

Apprenez à mettre à jour plusieurs documents à l'aide de la `updateMany()` méthode dans MongoDB.

MongoDB fournit les méthodes suivantes pour mettre à jour les documents existants dans une collection :

- [db.collection.updateOne\(\)](#) - Modifie un seul document dans une collection.
- [db.collection.updateMany\(\)](#) - Modifie un ou plusieurs documents dans une collection.

Dans la plupart des cas, il est recommandé d'utiliser la `updateMany()` méthode plutôt que la `updateOne()` méthode.

db.collection.updateMany()

Utilisez la `db.<collection>.updateMany()` méthode pour mettre à jour plusieurs documents correspondant aux critères de filtre spécifiés dans une collection.

Syntaxe:

`db.collection.updateMany(filter, document, options)`

Paramètres:

1. **filtre** : Les critères de sélection pour la mise à jour, identiques à la méthode [find\(\)](#) .
2. **document** : document ou pipeline contenant des modifications à appliquer.

3. Options : Facultatif. Peut contenir des options pour le comportement de mise à jour. Cela inclut upsert, writeConcern, classement, etc.

Dans la syntaxe ci-dessus, dbpointe vers la base de données actuelle, <collection>points est un nom de collection existant.

Pour illustrer l'opération de mise à jour, insérez les exemples de documents suivants dans la employeescollection.

Exemple de données

```
db.employees.insertMany([
  {
    _id:1,
    firstName: "John",
    lastName: "King",
    email: "john.king@abc.com",
    salary: 5000
  },
  {
    _id:2,
    firstName: "Sachin",
    lastName: "T",
    email: "sachin.t@abc.com",
    salary: 8000
  },
  {
    _id:3,
    firstName: "James",
    lastName: "Bond",
    email: "jamesb@abc.com",
    salary: 7500
  },
  {
    _id:4,
    firstName: "Steve",
    lastName: "J",
    email: "steve.j@abc.com",
    salary: 7000
  },
  {
    _id:5,
```

```

firstName: "Kapil",
lastName: "D",
email: "kapil.d@abc.com",
salary: 4500
},
{
_id:6,
firstName: "Amitabh",
lastName: "B",
email: "amitabh.b@abc.com",
salary: 7000
}
])

```

Ce qui suit modifie les documents correspondants à l'aide de la `updateMany()` méthode in `employees` collection.

Exemple : `updateMany()`

Copie

```
db.employees.updateMany({ salary:7000 }, { $set: { salary:8500 } })
```

Sortir

```
{
acknowledged: true,
insertedId: null,
matchedCount: 2,
modifiedCount: 2,
upsertedCount: 0
}
```

Dans l'exemple ci-dessus, le premier paramètre est le critère de filtre spécifié en tant que document, `{ salary:7000 }` indique que rechercher les documents dont `salary` sont `7000`. Le deuxième paramètre permet de spécifier les champs et les valeurs à modifier sur le document correspondant au `{<update-operator>: { field: value, field:value,... } }` format. Utilisez l'[opérateur de mise à jour](#) pour spécifier une action à effectuer. Ici, nous voulons définir la valeur des champs, utilisez donc `$set` l'opérateur pour spécifier les champs et les valeurs mises à jour au `{field:updated-value}` format. `{ $set: { salary:8500 } }` modifie les `salary` champs de tous les documents correspondants en `8500`.

Dans la sortie, `matchedCount` indique le nombre de documents correspondant aux critères et `modifiedCount` indique le nombre de documents mis à jour.

Maintenant, vérifiez s'il a mis à jour une valeur ou non en utilisant la `find()` méthode indiquée ci-dessous.

Vérifier le document mis à jour

Copie

`db.employees.find()`

Sortir

```
[  
  {  
    _id: 1,  
    firstName: 'John',  
    lastName: 'King',  
    email: 'john.king@abc.com',  
    salary: 5000  
  },  
  {  
    _id: 2,  
    firstName: 'Sachin',  
    lastName: 'T',  
    email: 'sachin.t@abc.com',  
    salary: 8000  
  },  
  {  
    _id: 3,  
    firstName: 'James',  
    lastName: 'Bond',  
    email: 'jamesb@abc.com',  
    salary: 7500  
  },  
  {  
    _id: 4,  
    firstName: 'Steve',  
    lastName: 'J',  
    email: 'steve.j@abc.com',  
    salary: 8500  
  },  
  {  
    _id: 5,  
    firstName: 'Kapil',  
    lastName: 'D',  
    email: 'kapil.d@abc.com',  
  }
```

```

    salary: 4500
},
{
  _id: 6,
  firstName: 'Amitabh',
  lastName: 'B',
  email: 'amitabh.b@abc.com',
  salary: 8500
}
]

```

La `updateMany()` méthode ajoute le champ spécifié s'il n'existe pas dans un document correspondant. Par exemple, ce qui suit ajoutera le `location` champ.

Exemple : `updateMany()`

Copie

```
db.employees.updateMany({firstName:"Steve"}, { $set: {location: "USA" }})
```

Sortir

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Exécutez la méthode suivante `find()` pour voir les données mises à jour.

Vérifier le document mis à jour

Copie

```
db.employees.find({firstName:"Steve"})
```

Sortir

```
{
  _id:4,
  firstName: "Steve",
  lastName: "J",
  email: "steve.j@abc.com",
  salary: 8500,
  location: "USA"
}
```

Si vous spécifiez un critère de filtre vide {}, tous les documents seront mis à jour. Ce qui suit mettra à jour ou ajoutera locationun champ dans tous les documents.

Exemple : updateMany()

Copie

```
db.employees.updateMany({}, { $set: {location: "USA" }})
```

Sortir

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 6,
  modifiedCount: 6,
  upsertedCount: 0
}
```

Utilisez l' \$incopérateur de mise à jour pour augmenter la valeur du champ du montant spécifié. Ce qui suit augmente le salarydont 500le salaire est 8500.

Exemple : Opérateur \$inc

Copie

```
db.employees.updateMany({salary:8500}, { $inc: {salary: 500 }})
```

Sortir

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
```

Mettre à jour plusieurs champs

Vous pouvez également spécifier plusieurs champs à mettre à jour. Email Les mises à jour et les champs suivants lastName.

Exemple : Mettre à jour plusieurs champs

Copie

```
db.employees.updateMany({_id:2}, { $set: {lastName:"Tendulkar",
email:"sachin.tendulkar@abc.com" }})
```

Sortir

```
{
```

```

acknowledged: true,
insertedId: null,
matchedCount: 1,
modifiedCount: 1,
upsertedCount: 0
}

```

Exécutez la méthode suivante `find()` pour voir les données mises à jour.

Vérifier le document mis à jour

Copie

```
db.employees.find({_id:2})
```

Sortir

```
{
  _id:2,
  firstName: "Sachin",
  lastName: "Tendulkar",
  email: "sachin.tendulkar@abc.com",
  salary: 8000
}
```

La `updateMany()` méthode ne met à jour aucun document si aucun document correspondant n'est trouvé. Par exemple, les éléments suivants ne mettront à jour aucun document.

Exemple : `updateMany()`

Copie

```
db.employees.updateMany({salary:100}, { $set: {salary:200}})
```

Sortir

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0}
```

Upsert - Ajouter s'il n'existe pas

Spécifiez `{upsert:true}` comme troisième paramètre dans la `UpdateMany()` méthode. Le `upsert:true` ajoute un nouveau document si le document correspondant est introuvable.

Exemple : Upsert

Copie

```
db.employees.updateMany({firstName:"Heer"}, { $set: {lastName:"Patel", salary:2000}})
```

Sortir

```
{
  acknowledged: true,
  insertedId: ObjectId("6172a2e9ce7d5ca09d6ab082"),
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1
}
```

Dans l'exemple ci-dessus, MongoDB ajoute un nouveau document avec new `_id`, car il ne trouve pas de document avec le `firstName:"Heer"`.

Opérateurs de mise à jour

Le tableau suivant répertorie les opérateurs de mise à jour qui peuvent être utilisés avec les méthodes `updateOne()` et `updateMany()`.

Méthode	Description
<code>\$currentDate</code>	Définit la valeur d'un champ à la date actuelle, soit en tant que date, soit en tant qu'horodatage.
<code>\$inc</code>	Incrémente la valeur du champ du montant spécifié.
<code>\$min</code>	Ne met à jour le champ que si la valeur spécifiée est inférieure à la valeur du champ existant.
<code>\$max</code>	Ne met à jour le champ que si la valeur spécifiée est supérieure à la valeur du champ existant.
<code>\$mul</code>	Multiplie la valeur du champ par le montant spécifié.
<code>\$ renommer</code>	Renomme un champ.
<code>\$ ensemble</code>	Définit la valeur d'un champ dans un document.
<code>\$setOnInsert</code>	Définit la valeur d'un champ si une mise à jour entraîne l'insertion d'un document. N'a aucun effet sur les opérations de mise à jour qui modifient les documents existants.

Méthode	Description
\$unset	Supprime le champ spécifié d'un document.

MongoDB : Mettre à jour les tableaux dans les documents

Découvrez comment mettre à jour les champs de tableau dans les documents des collections MongoDB.

Vous pouvez utiliser les méthodes [updateOne\(\)](#) ou [updateMany\(\)](#) pour ajouter, mettre à jour ou supprimer des éléments de tableau en fonction des critères spécifiés. Il est recommandé d'utiliser la [updateMany\(\)](#) méthode pour mettre à jour plusieurs tableaux dans une collection.

Pour illustrer l'opération de mise à jour, insérez les exemples de documents suivants dans la `employees` collection.

Exemple de données

Copie

```
db.employees.insertMany([
  {
    _id:1,
    firstName: "John",
    lastName: "King",
    email: "john.king@abc.com",
    salary: 5000,
    skills: [ "Angular", "React", "MongoDB" ]
  },
  {
    _id:2,
    firstName: "Sachin",
    lastName: "T",
    email: "sachin.t@abc.com",
    salary: 8000,
    skills: [ "Accounting", "Tax" ]
  },
  {
    _id:3,
```

```

firstName: "James",
lastName: "Bond",
email: "jamesb@abc.com",
salary: 7500,
skills: [ "Sales", "Marketing" ]
},
{
  _id:4,
  firstName: "Steve",
  lastName: "J",
  email: "steve.j@abc.com",
  salary: 7000,
  skills: [ "Mac", "Marketing", "Product Design" ]
},
{
  _id:5,
  firstName: "Kapil",
  lastName: "D",
  email: "kapil.d@abc.com",
  salary: 4500,
  skills: [ "Accounting", "Tax", "Sales" ]
},
{
  _id:6,
  firstName: "Amitabh",
  lastName: "B",
  email: "amitabh.b@abc.com",
  salary: 7000,
  skills: [ "Marketing", "Tax" ]
}
])

```

Écraser les tableaux

L' `$set` opérateur écrase le tableau spécifié au lieu d'ajouter, de supprimer et de mettre à jour des éléments de tableau.

Exemple : Écraser le tableau

Copie

`db.employees.updateMany({ _id:5 }, { $set: { skills:["Sales Tax"] } })`

Sortir

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Dans l'exemple ci-dessus, `{$set:{ skills:["Sales Tax"]}}` remplace un tableau existant pour `{_id:5}`.

Vérifier le document mis à jour

Copie

```
db.employees.find({_id:5})
```

Sortir

```
{
  _id: 5,
  firstName: 'Kapil',
  lastName: 'D',
  email: 'kapil.d@abc.com',
  salary: 4500,
  skills: [ 'Sales Tax' ],
  department: { name: 'Finance' },
  location: 'USA'
}
```

Mettre à jour les éléments du tableau

Utilisez les [opérateurs de tableau](#) pour mettre à jour un ou plusieurs éléments de tableaux dans MongoDB.

Ce qui suit mettra à jour "Marketing" en "Public Speaking" dans le `skills` champ tableau de tous les documents.

Exemple : Mettre à jour les éléments du tableau

Copie

```
db.employees.updateMany(
  {skills:"Marketing"},
  {$set:{ "skills.$":"Public Speaking" }})
```

Sortir

```
{
```

```

acknowledged: true,
insertedId: null,
matchedCount: 2,
modifiedCount: 2,
upsertedCount: 0
}

```

Dans l'exemple ci-dessus, `{skills:"Marketing"}` spécifie les critères pour trouver tous les documents où le `skillstableau` contient "Marketing"element.

Le deuxième paramètre `{$set: { "skills.$": "Public Speaking" } }` spécifie la valeur à mettre à jour à l'aide `$set`de l'opérateur. Le `{ "skills.$": "Public Speaking" }` spécifie de mettre à jour l'élément sur "Public Speaking". `$` est un opérateur de tableau qui agit comme un espace réservé pour la première correspondance du document de requête de mise à jour.

Exemple : Mettre à jour les éléments du tableau

Copie

```

db.employees.updateMany(
  {},
  { $set: { "skills.$[element]": "GST" } },
  { arrayFilters: [ { element: "Tax" } ] }
)

```

Sortir

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 6,
  modifiedCount: 2,
  upsertedCount: 0
}

```

Dans l'exemple ci-dessus, `{ $set: { "skills.$[element]": "GST" } }, { arrayFilters: [{ element: "Tax" }] }` met à jour le `skillstableau` s'il contient l'élément "Tax", puis le met à jour en "GST". `{ "skills.$[element]": "GST" }` spécifie de mettre à jour l'élément en "GST" et `arrayFilters`spécifie les critères pour les éléments du tableau. Le `{ arrayFilters: [{ element: "Tax" }] }`spécifie que l'élément de tableau de recherche dont la valeur est "Taxe". Ainsi, la `updateMany()`méthode mettra à jour les éléments du tableau avec la valeur spécifiée par `$set`et pour les éléments correspondants spécifiés par `arrayFilters`.

Ajouter un nouvel élément aux tableaux

Utilisez l' `$push` opérateur de tableau pour ajouter de nouveaux éléments aux tableaux. Ce qui suit ajoutera "Sports" un élément dans tous les tableaux.

Exemple : Ajouter des éléments de tableau

Copie

```
db.employees.updateMany(
  {},
  {$push:{"skills":"Sports"}}) // add "Sports" to all arrays
```

```
db.employees.updateMany(
  {_id:3},
  {$push:{"skills":"Sports"}}) // add "Sports" element to skills array where _id:3
```

Utilisez l' `$each` opérateur pour spécifier plusieurs éléments qui doivent être ajoutés dans les tableaux.

Exemple : Ajouter des éléments de tableau

Copie

```
db.employees.updateMany(
  {},
  {$push:{"skills":{$each:["Sports","Acting"]}}}) // adds "Sports" and "Acting" to all arrays
```

Dans l'exemple ci-dessus, `{$each:["Sports","Acting"]}` spécifie un tableau pour ajouter plusieurs éléments.

Utilisez `$addToSet` l'opérateur pour ajouter un élément s'il n'existe pas déjà.

Ce qui suit s'ajoutera "GST" au `skills` tableau dans tous les documents s'il n'existe pas.

Exemple : Ajouter un élément s'il n'existe pas

Copie

```
db.employees.updateMany(
  {},
  {$addToSet: {"skills":"GST"}}) // adds "GST" to all arrays if not exist
```

Supprimer le premier ou le dernier élément des tableaux

Utilisez l' `$pop` opérateur pour supprimer le premier ou le dernier élément des tableaux. Spécifiez 1 pour supprimer le dernier élément et -1 pour supprimer le premier élément.

Exemple : Supprimer un élément de tableau

Copie

```
db.employees.updateMany(
  {},
  {$pop: { "skills": 1 } }) // removes the last element
```

```
db.employees.updateMany(
  {},
  {$pop: { "skills": -1 } }) // removes the first element
```

```
db.employees.updateMany(
  {},
  {$pull: { "skills": "GST" }}) // removes "GST"
```

MongoDB : Supprimer des documents dans une collection

MongoDB fournit les méthodes suivantes pour supprimer un ou plusieurs documents d'une collection.

- [db.collection.deleteOne\(\)](#) - Supprime le premier document correspondant dans une collection.
- [db.collection.deleteMany\(\)](#) - Supprime tous les documents correspondants dans une collection.

db.collection.deleteOne()

Utilisez la `db.<collection>.deleteOne()` méthode pour supprimer les premiers documents qui correspondent aux critères de filtre spécifiés dans une collection.

Syntaxe:

`db.collection.deleteOne(filter, options)`

Paramètres:

1. filtre : Les critères de sélection pour la mise à jour, identiques à la méthode [find\(\)](#) .
2. Options : Facultatif. Peut contenir des options pour le comportement de mise à jour. Il inclut les paramètres writeConcern, collation et hint.

Dans la syntaxe ci-dessus, `db` pointe vers la base de données actuelle, `<collection>` pointe vers un nom de collection existant.

Pour illustrer l'opération de suppression, insérez les exemples de documents suivants dans la `employees` collection.

Exemple de données

Copie

```
db.employees.insertMany([
  {
    _id:1,
    firstName: "John",
    lastName: "King",
    email: "john.king@abc.com",
    salary: 5000
  },
  {
    _id:2,
    firstName: "Sachin",
    lastName: "T",
    email: "sachin.t@abc.com",
    salary: 8000
  },
  {
    _id:3,
    firstName: "James",
    lastName: "Bond",
    email: "jamesb@abc.com",
    salary: 7500
  },
  {
    _id:4,
    firstName: "Steve",
    lastName: "J",
    email: "steve.j@abc.com",
    salary: 7000
  },
  {
    _id:5,
    firstName: "Kapil",
    lastName: "D",
    email: "kapil.d@abc.com",
    salary: 4500
  },
  {
    _id:6,
    firstName: "Rajesh",
    lastName: "Kumar",
    email: "rajesh.kumar@abc.com",
    salary: 6000
  }
])
```

```

_id:6,
firstName: "Amitabh",
lastName: "B",
email: "amitabh.b@abc.com",
salary: 7000
}
])

```

Ce qui suit supprime un document de la `employeescollection`.

Exemple : Supprimer un seul document à l'aide de `deleteOne()`

Copie

```
db.employees.deleteOne({ salary:7000 })
```

Sortir

```
{ acknowledged: true, deletedCount: 1 }
```

La commande ci-dessus supprime le premier document correspondant même si plusieurs documents correspondent aux critères spécifiés.

`deleteMany()`

Utilisez la `db.<collection>.deleteMany()` méthode pour supprimer tous les documents qui correspondent aux critères de filtre spécifiés dans une collection.

Syntaxe:

```
db.collection.deleteMany(filter, options)
```

Paramètres:

1. **filtre** : Les critères de sélection pour la mise à jour, identiques à la méthode [find\(\)](#) .
2. **Options** : Facultatif. Peut contenir des options pour le comportement de mise à jour. Il inclut `writeConcern` et `collation`, et les paramètres d'indication.

Ce qui suit supprime tous les documents de la `employeescollection` qui correspondent aux critères spécifiés.

Exemple : `deleteMany()`

Copie

```
db.employees.deleteMany({ salary:7000 })
```

Sortir

```
{ acknowledged: true, deletedCount: 2 }
```

Ce qui suit supprime tous les documents où salaryest inférieur à 7000.

Exemple : deleteMany()

Copie

```
db.employees.deleteMany({ salary: { $lt:7000 } })
```

Sortir

```
{ acknowledged: true, deletedCount: 2 }
```

Si vous spécifiez un critère vide, tous les documents d'une collection seront supprimés.

Exemple : Supprimer tous les documents

Copie

```
db.employees.deleteMany({ }) //deletes all documents
```

Sortir

```
{ acknowledged: true, deletedCount: 6 }
```

CHAPITRE 4 : AGRÉGATION DANS MONGODB

L'agrégation est le processus de sélection des données d'une collection dans MongoDB. Il traite plusieurs documents et renvoie des résultats calculés.

Utilisez l'agrégation pour regrouper les valeurs de plusieurs documents ou effectuez des opérations sur les données groupées pour renvoyer un seul résultat.

Les opérations d'agrégation peuvent être effectuées de deux manières :

1. Utilisation du pipeline d'agrégation.
2. Utilisation de méthodes d'agrégation à usage unique : [db.collection.estimatedDocumentCount\(\)](#), [db.collection.count\(\)](#) et [db.collection.distinct\(\)](#).

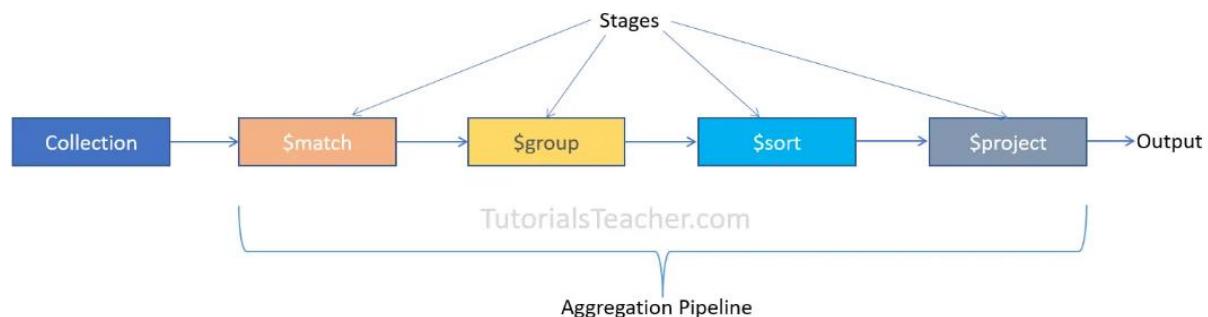
Pipelines d'agrégation

Le pipeline d'agrégation est un tableau d'une ou plusieurs étapes passées dans la méthode `db.aggregate()` ou `db.collection.aggregate()`.

```
db.collection.aggregate([ {stage1}, {stage2}, {stage3}...])
```

Le cadre d'agrégation traite le pipeline d'étapes sur les données de collecte et vous donne une sortie sous la forme dont vous avez besoin.

Chaque étape reçoit la sortie de l'étape précédente, traite les données plus avant et les envoie à l'étape suivante en tant que données d'entrée. Le pipeline d'agrégation exécuté sur le serveur peut tirer parti des index



Voyons comment utiliser différentes étapes sur la collection `employees` suivante.

Exemple de données

Copie

```
db.employees.insertMany([
```

```
{}
```

```

_id:1,
firstName: "John",
lastName: "King",
gender:'male',
email: "john.king@abc.com",
salary: 5000,
department: {
    "name":"HR"
},
{
_id:2,
firstName: "Sachin",
lastName: "T",
gender:'male',
email: "sachin.t@abc.com",
salary: 8000,
department: {
    "name":"Finance"
},
{
_id:3,
firstName: "James",
lastName: "Bond",
gender:'male',
email: "jamesb@abc.com",
salary: 7500,
department: {
    "name":"Marketing"
},
{
_id:4,
firstName: "Rosy",
lastName: "Brown",
gender:'female',
email: "rosyb@abc.com",
salary: 5000,
department: {
    "name":"HR"
},
{
_id:5,

```

```

firstName: "Kapil",
lastName: "D",
gender:'male',
email: "kapil.d@abc.com",
salary: 4500,
department: {
    "name":"Finance"
}

},
{
    _id:6,
    firstName: "Amitabh",
    lastName: "B",
    gender:'male',
    email: "amitabh.b@abc.com",
    salary: 7000,
    department: {
        "name":"Marketing"
    }
}
])

```

Étape \$match

L' `$match` étape est généralement la première étape pour sélectionner uniquement les documents correspondants d'une collection. Elle est équivalente à la méthode [Find\(\)](#). L'exemple suivant illustre un pipeline d'agrégation avec une seule `$match` étape.

Exemple : Étape `$match`

Copie

```
db.employees.aggregate([ { $match:{ gender: 'female'} } ])
```

Dans l'exemple ci-dessus, l'étape `$match` est spécifiée en tant que document `[$match:{ gender: 'female'}]` dans un tableau. Il renverra tous les documents `gender:'female'` déposés.

Sortir

```
[
{
    _id: 4,
    firstName: 'Rosy',
    lastName: 'Brown',
    gender: 'female',
    email: 'rosyb@abc.com',
    salary: 5000,
    department: { name: 'HR' }
```

```
}
```

```
]
```

L'étape \$match de la méthode aggregate() donne le même résultat que la find() méthode. Le **db.persons.find({ gender: 'female' })** renvoie les mêmes données que ci-dessus.

\$étape de groupe

Utilisez l'étape \$group pour regrouper les documents d'entrée par l' `_id` expression spécifiée et renvoie un seul document contenant les valeurs cumulées pour chaque groupe distinct. Prenons l'exemple suivant.

Exemple : étape \$groupe

Copie

```
db.employees.aggregate([
  { $group:{ _id:'$department.name' } }
])
```

Sortir

```
[ { _id: 'Marketing' }, { _id: 'HR' }, { _id: 'Finance' } ]
```

Dans l'exemple ci-dessus, seule l' `$group` étape est spécifiée dans le tableau de pipeline. Le champ utilise `$group _id` pour calculer les valeurs cumulées pour tous les documents d'entrée dans leur ensemble. L'expression crée `{ _id:'$department.name'}` le groupe distinct sur le champ `$department.name`. Comme nous ne calculons aucune valeur cumulée, il renvoie les valeurs distinctes de `$department.name`, comme indiqué ci-dessous.

Maintenant, calculons les valeurs cumulées pour chaque groupe. Ce qui suit calcule le nombre d'employés dans chaque département.

Exemple : Obtenir les valeurs cumulées

```
db.employees.aggregate([
  { $group:{ _id:'$department.name', totalEmployees: { $sum:1 } } }
])
```

Sortir

```
[
  { _id: 'Marketing', totalEmployees: 2 },
  { _id: 'HR', totalEmployees: 2 },
  { _id: 'Finance', totalEmployees: 2 }
]
```

Dans l'exemple ci-dessus, nous créons des groupes distincts à l'aide `_id:'$department.name'` de l'expression. Dans la deuxième expression `totalEmployees: { $sum:1 }`, `totalEmployees` est un champ qui sera inclus dans la sortie et `{ $sum:1 }` est une expression d'accumulateur où `$sum` est un [opérateur d'accumulateur](#) qui renvoie une somme de valeurs numériques. Ici, `{ $sum:1 }` ajoute 1 pour chaque document appartenant au même groupe.

Le pipeline d'agrégation suivant contient deux étapes.

Exemple : \$match et \$group

Copie

```
db.employees.aggregate([
  { $match: { gender:'male'} },
  { $group:{ _id:$department.name', totalEmployees: { $sum:1 } } }
])
Sortir
[
  { _id: 'Marketing', totalEmployees: 2 },
  { _id: 'HR', totalEmployees: 1 },
  { _id: 'Finance', totalEmployees: 2 }
]
```

Dans l'exemple ci-dessus, la première étape sélectionne tous les employés masculins et les transmet comme entrée à la deuxième étape \$group comme entrée. Ainsi, la sortie calcule la somme de tous les employés masculins.

Ce qui suit calcule la somme des salaires de tous les employés masculins du même département.

Exemple : obtenir la somme des champs

Copie

```
db.employees.aggregate([
  { $match:{ gender:'male'} },
  { $group:{ _id:{ deptName:$department.name'}, totalSalaries: { $sum:'$salary'} } }
])
Sortir
[
  { _id: 'Finance', totalSalaries: 12500 },
  { _id: 'HR', totalSalaries: 10000 },
  { _id: 'Marketing', totalSalaries: 14500 }
]
```

Dans l'exemple ci-dessus, { \$match:{ gender:'male'} } renvoie tous les employés de sexe masculin. Dans l' \$group étape, une expression d'accumulateur totalSalaries: { \$sum:'\$salary'} résume le champ numérique salary et l'inclut comme totalSalaries dans la sortie de chaque groupe.

\$trier l'étape

L' \$sort étape est utilisée pour trier les documents en fonction du champ spécifié dans l'ordre croissant ou décroissant. Ce qui suit trie tous les employés de sexe masculin.

Exemple : Trier des documents

Copie

```

db.employees.aggregate([
  { $match: { gender:'male'} },
  { $sort: { firstName:1 } }
])
Sortir
[
  {
    _id: 6,
    firstName: 'Amitabh',
    lastName: 'B',
    gender: 'male',
    email: 'amitabh.b@abc.com',
    salary: 7000,
    department: { name: 'Marketing' }
  },
  {
    _id: 3,
    firstName: 'James',
    lastName: 'Bond',
    gender: 'male',
    email: 'jamesb@abc.com',
    salary: 7500,
    department: { name: 'Marketing' }
  },
  {
    _id: 1,
    firstName: 'John',
    lastName: 'King',
    gender: 'male',
    email: 'john.king@abc.com',
    salary: 5000,
    department: { name: 'HR' }
  },
  {
    _id: 5,
    firstName: 'Kapil',
    lastName: 'D',
    gender: 'male',
    email: 'kapil.d@abc.com',
    salary: 4500,
    department: { name: 'Finance' }
  },
  {
    _id: 2,
    firstName: 'Sachin',
    lastName: 'T',
  }
]
  
```

```

gender: 'male',
email: 'sachin.t@abc.com',
salary: 8000,
department: { name: 'Finance' }
}
]

```

Dans l'exemple ci-dessus, l' \$match étape renvoie tous les employés masculins et les transmet à l'étape suivante \$sort. L' { \$sort:{ firstName:1 } } expression trie les documents d'entrée par firstName champ dans l'ordre croissant. 1 indique l'ordre croissant et -1 indique l'ordre décroissant.

Le pipeline suivant contient trois étapes pour trier les documents groupés.

Exemple : Trier des données groupées

Copie

```

db.employees.aggregate([
  { $match:{ gender:'male'} },
  { $group:{ _id:{ deptName:$department.name }, totalEmployees: { $sum:1 } } },
  { $sort:{ deptName:1 } }
])

```

Sortir

```
[
  { _id: { deptName: 'Finance' }, totalEmployees: 2 },
  { _id: { deptName: 'HR' }, totalEmployees: 1 },
  { _id: { deptName: 'Marketing' }, totalEmployees: 2 }
```

CHAPITRE 5 : RELATIONS DANS MONGODB : UN A UN, UN A PLUSIEURS, PLUSIEURS A PLUSIEURS

Dans les bases de données RDBMS telles que SQL Server, les relations entre les tables peuvent être un-à-un, un-à-plusieurs et plusieurs-à-plusieurs.

Dans MongoDB, les relations un-à-un, un-à-plusieurs et plusieurs-à-plusieurs peuvent être implémentées de deux manières :

1. Utilisation de documents intégrés
2. Utiliser la référence de documents d'une autre collection

Mettre en œuvre la relation à l'aide du document intégré

Vous pouvez inclure des données associées en tant que documents intégrés. Par exemple, vous pouvez inclure une adresse en tant que document intégré, comme indiqué ci-dessous.

Exemple : Relation à l'aide d'un document intégré

Copie

```
db.employee.insertOne({
  _id: ObjectId("32521df3f4948bd2f54218"),
  firstName: "John",
  lastName: "King",
  email: "john.king@abc.com",
  salary: "33000",
  DoB: new Date('Mar 24, 2011'),
  address: {
    street: "Upper Street",
    house: "No 1",
    city: "New York",
    country: "USA"
  }
})
```

Implémenter la relation à l'aide de la référence

Une autre façon d'implémenter des relations consiste à utiliser la référence du champ clé primaire des documents d'une autre collection.

Par exemple, créez `address` une collection et utilisez- `_id` la comme référence d'un document de la `employee` collection.

Exemple : implémenter une relation un-à-un à l'aide d'une référence

Copie
`db.address.insertOne({
 _id: 101,
 street:"Upper Street",
 house:"No 1",
 city:"New York",
 country:"USA"
})`

`db.employee.insertOne({
 firstName: "John",
 lastName: "King",
 email: "john.king@abc.com",
 salary: "33000",
 DoB: new Date('Mar 24, 2011'),
 address: 101
})`

Dans l'exemple ci-dessus, la relation entre `employee` et `address` collection est implémentée à l'aide des identifiants de référence. Un document de la collection `employee` contient un `address` champ qui a la valeur d'un existant `_id` dans la `address` collection. Il forme des relations un à un.

Remarque : Vous pouvez référencer n'importe quel champ pour la relation, mais il est recommandé d'utiliser le champ de clé primaire unique pour éviter les erreurs.

Vous pouvez récupérer les données associées en deux étapes. Ce qui suit récupère l'adresse d'un employé.

Exemple : Rechercher des documents associés

`var addrId = db.employee.findOne({firstName:'John'}).address;`

`db.address.findOne({_id:addrId});`

Dans l'exemple ci-dessus, récupérez le `addrId` champ d'un employé, puis recherchez le document d'adresse en utilisant `addrId`.

Utilisez l'étape de pipeline d'agrégation [\\$lookup](#) pour rechercher les données associées de la collection, comme indiqué ci-dessous.

Exemple : \$lookup pour obtenir des documents associés

Copie

`db.employee.aggregate([{$lookup:{from:'address',localField:'address',foreignField:'_id',as:'addr'}}])`

Sortir

[

```
{  
  _id: ObjectId("617a75c013dceca5c350d52f"),  
  firstName: 'John',  
  lastName: 'King',  
  email: 'john.king@abc.com',  
  salary: '33000',  
  DoB: ISODate("2011-03-23T18:30:00.000Z"),  
  address: 101,  
  addr: [  
    {  
      _id: 101,  
      street: 'Upper Street',  
      house: 'No 1',  
      city: 'New York',  
      country: 'USA'  
    }  
  ]  
}
```

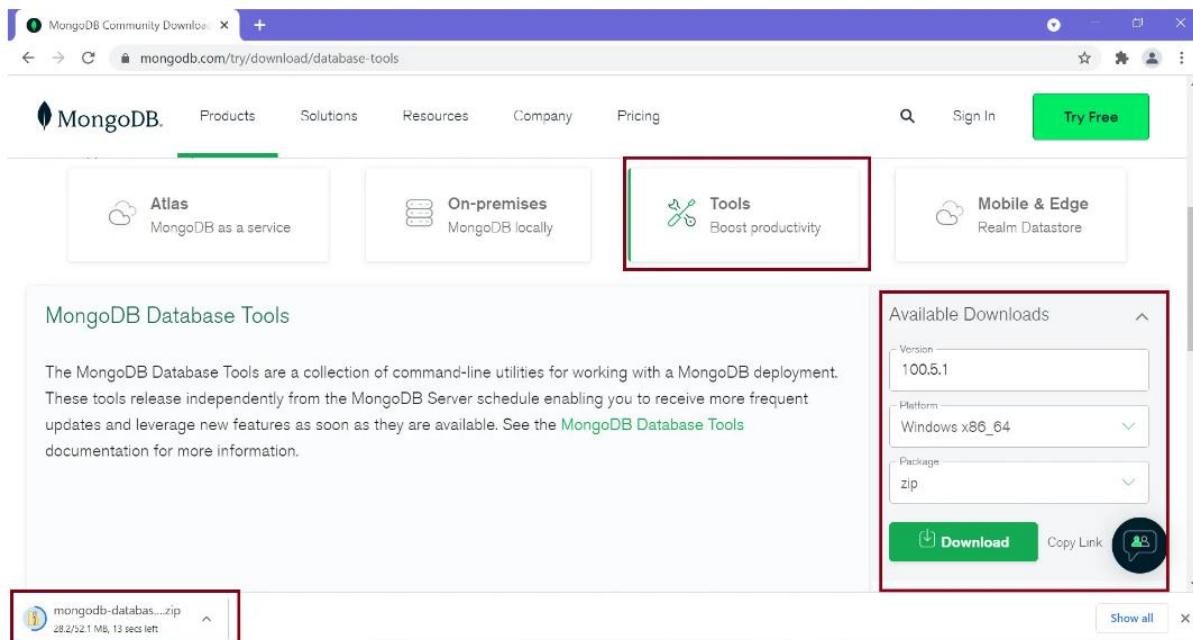
De la même manière, vous pouvez implémenter des relations un-à-plusieurs et plusieurs-à-plusieurs dans MongoDB.

CHAPITRE 6 : IMPORTER DES DONNEES DANS MONGODB A L'AIDE DE MONGOIMPORT

Ici, vous allez apprendre à importer des données JSON ou un fichier CSV dans une collection dans MongoDB.

Utilisez `mongoimport` la commande pour importer des données dans une collection. Vous devez avoir installé les outils de base de données MongoDB pour utiliser la `mongoimport` commande.

Pour installer les outils de base de données, visitez Outils de base de données et téléchargez le fichier zip pour votre plate-forme.



Maintenant, extrayez et copiez tous les fichiers .exe et collez-les dans le dossier bin de MongoDB. Sous Windows, c'est `C:\Program Files\MongoDB\Server\<version>\bin` dossier.

```

    bsondump.exe
    mongodump.exe
    mongoexport.exe
    mongofiles.exe
    mongoimport.exe
    mongorestore.exe
    mongostat.exe
    mongotop.exe
  
```

Maintenant, ouvrez le terminal ou l'invite de commande et accédez à l'emplacement où vous avez le fichier JSON à importer afin de ne pas avoir à spécifier le chemin complet.

Voici la commande mongoimport.

```
mongoimport --db nom_base_de_données --collection nom_collection ^
--authenticationDatabase admin --username <utilisateur> --password <mot de passe> ^
--file chemin_fichier
```

Maintenant, exécutez la commande suivante pour importer des données du D:\MyData\employeesdata.jsonfichier à employeesla collection

```
D:\MyData> mongoimport --db test --collection employee --file employeedata.json --
```

jsonArray

La commande ci-dessus importera des données dans la employeescollection de la testbase de données. Notez que cela --jsonArrayindique que les données d'un fichier contiennent dans un tableau.

Importer des données à partir d'un fichier CSV

Considérez que vous avez D:\employeesdata.csvun fichier que vous souhaitez importer dans une nouvelle employeecollection. Exécutez la commande suivante pour importer des données à partir du fichier CSV.

```
D:\MyData> mongoimport --db test --collection employeedata --type csv --file
```

employee.csv --fields _id,firstName,lastName

L' --fieldsoption indique les noms de champs à utiliser pour chaque colonne du fichier CSV. Si un fichier contient la ligne d'en-tête qui doit être utilisée comme nom de champ, utilisez --headerlineoption au lieu de --fields. La commande ci-dessus insérera toutes les données dans employeesla collection, comme indiqué ci-dessous.

```
test> db.employees.find()  
[  
  { _id : 2, firstName : 'bill', lastName : 'gates' },  
  { _id : 1, prénom : 'steve', nom : 'emplois' },  
  { _id : 3, prénom : 'james', nom : 'lien' }  
]
```

PARTIE 2: DATA WAREHOUSING ET BIG DATA