

How To:

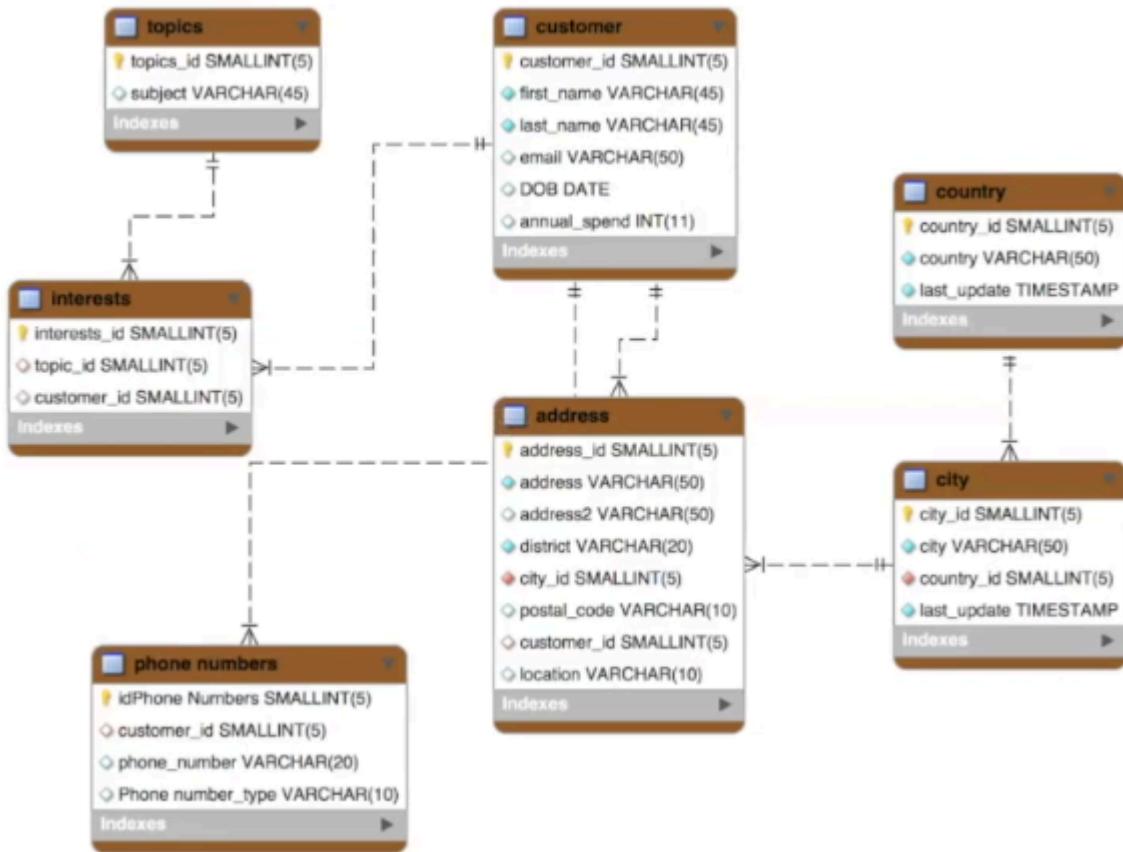


Cédric FONTAINE

Préambule	4
Retour sur les SGBD « traditionnels »	4
Le paradigme MongoDB	5
JSON	5
Bases de données MongoDB : Collections & documents	5
Conception des schémas	6
Intégration	6
Référencement	9
Bucketing	9
Chapitre 0 - Installation	10
MongoDB	10
Ajouter Mongo au PATH	10
Compass	13
MongoDB Atlas	14
Chapitre 1 - Introduction	18
Le principe champ-valeur	18
Exemple de type composite : Les données géospatiales	18
Comprendre l'analyse de Schémas sur Compass	19
Chapitre 2 - CRUD	22
Préparation	22
Read	22
Create	26
Update	27
Delete	31
Chapitre 3 - Query language	32
Les filtres d'égalité	32
Les filtres de portée	32
Les principaux opérateurs	33
Chapitre 4 - Procédures stockées	36
Création	36
Utilisation	36
Annexes	37
Sources	37
Documentation	37
Bréviaire - CRUD	37
Ressources complémentaires	38

Préambule

Retour sur les SGBD « traditionnels »



Idée primaire de la normalisation des données par les schémas relationnels :

Structurer les données en fonction des relations qu'elles ont entre elles. On précise exactement les champs et leur domaine de définition avant d'y insérer des données.

Avantages : Mise en place de contraintes permettant de vérifier la cohérence des données, limite la redondance et la perte d'intégrité.

Inconvénients : Des schémas pouvant être extrêmement complexes (des centaines de relations interconnectées).

Dette technique : malgré une méthode générique de conception des systèmes d'information, tous les SGBDR ne fonctionnent pas de la même manière. Toutes les optimisations de performances, procédures stockées etc., sont dépendantes du système de gestion utilisé.

Le paradigme MongoDB

MongoDB est un **SaaS (Software as a Service)** **noSQL (not only SQL)** qui offre un système **performant et scalable** (qui ne souffre pas de la taille des données stockées), orienté **documents**.

« Les données utilisées en même temps sont stockées au même endroit. »

« Le modèle relationnel a été conçu à la vue de la donnée, et le modèle document, à partir de son usage applicatif. »

JSON

Le **JSON (JavaScript Object Notation)** est un format de structuration de données basé sur le JavaScript. Ces données peuvent être des **nombres**, des **chaînes de caractère**, des **booléens**, la valeur **null**, ou bien des **tableaux** et d'autres **objets** (les champs peuvent être multi-valués). MongoDB utilise cette syntaxe pour stocker ses documents, mais dans un format sérialisé appelé **BSON (Binary JSON)**. Ce format léger et fluide supporte 17 types dits “composites”, c'est à dire créés à partir d'un ensemble de types natifs permettant un traitement et usage beaucoup plus poussé des données.

Note : On ne parle pas de documents au sens bureautique, mais bien d'une façon d'enregistrer des informations.

Bases de données MongoDB : Collections & documents

Une base de données contient des collections, qui contiennent elles-mêmes des documents. On forme un identifiant à partir du nom de sa base de données suivi de celui de la collection : `databaseName.collectionName`.

On pourrait comparer une collection à une table, et les documents aux tuples de celle-ci. Un document est donc un ensemble de paires “champ - valeur” représentant une entité. Il est principalement identifié avec un champ “`_id`” unique généré automatiquement ou renseigné à la création du document.

Chaque document peut contenir des champs totalement indépendants, ce qui rend Mongo plus souple que les SGBDR standards.

Conception des schémas

MongoDB n'est **pas totalement dénué de schémas** (*schemaless*), même s'il est beaucoup moins contraignant, puisqu'un schéma n'est pas défini **à priori**. Cela n'empêche pas d'établir des règles de validations

Il y a 4 questions à aborder pour bien construire sa base de données et éviter les problèmes de performance :

- Quelles-sont les modalités d'accès ?
(Unitairement, par agrégation ? Les modalités qui poussent généralement à dénormaliser une base de données relationnelle)
- Quel-est le ratio entre le nombre de lectures et le nombre d'écritures ?
- Quelle-est la taille prévue d'un document ?
(Un document allant jusqu'à 16 Mo, ce qui a une influence sur le traitement de l'information)
- Quelles-sont les cardinalités ?
(Combien de types seront traités, combien de sous-documents seront présents ? etc.)

Intégration

Si l'on prend ces 2 collections.

Collection Clients :

```
{  
  "_id": 1,  
  "lastname": "Doe",  
  "firstname": "Jane",  
  "city": "Paris"  
}  
  
{  
  "_id": 2,  
  "lastname": "Doe",  
  "firstname": "John",  
  "city": "Paris"  
}
```

Collection Cars :

```
{  
  "model": "Peugeot 207",  
  "year": 2013,  
  "color": "White",  
  "value": 1500.00,  
  "owner": 1  
}  
  
{  
  "model": "Dacia Duster",  
  "year": 2016,  
  "color": "Black",  
  "value": 15000.00,  
  "owner": 1  
}
```

Il apparaît évident que chaque voiture appartient à un client. On est dans une logique relationnelle où le champ **owner** est une référence à une personne (similaire à une clé étrangère pointant vers une autre table).

Il est tout à fait envisageable de garder un tel modèle s'il répond aux besoins de l'application, par exemple si l'on a envie de traiter les voitures séparément. En revanche, s'il s'agit d'un registre ayant pour but le traitement des clients uniquement, on peut tout à fait incruster (on parle alors *d'intégration de documents*) les documents voitures dans le document client.

La force de MongoDB réside dans la possibilité de stocker la totalité de ces informations dans un seul document BSON :

<pre>{ "_id": 1, "lastname": "Doe", "firstname": "Jane", "city": "Paris", "cars": [{ "model": "Peugeot 207", "year": 2013, "color": "White", "value": 1500.00, "owner": 1 }, { "model": "Dacia Duster", "year": 2016, "color": "Black", "value": 15000.00, "owner": 1 }] }</pre>	<pre>{ "_id": 2, "lastname": "Doe", "firstname": "John", "city": "Paris" } // Notons que ce client ne possède pas de voitures, et l'absence même du champ "cars" dans ce document.</pre>
--	--

Autre exemple, prenons l'entité suivante :

_id	lastname	firstname	age	email	phone	
					home	555-505-4341
[...]	Doe	Jane	33	jane.doe@contoso.com	cellphone	555-505-6864

Si nous voulions l'enregistrer dans une base de données relationnelle, nous aurions besoin d'au moins 2 relations :

```
Client [idPers, lastname, firstname, age, email]
Phone [idLine, type, number, #idPers]
```

Et pour répondre à la question :

Quels-sont les numéros de téléphone de Jane Doe, et leur type ?

```
SELECT type, number FROM DataBase.Client NATURAL JOIN DataBase.Phone  
WHERE lastname = 'Doe' AND firstname = 'Jane';
```

Une jointure nous renverrait le résultat suivant :

type	number
home	555-505-4341
cellphone	555-505-6864

1. Récupération du tuple de Jane Doe dans la table Client.
2. Récupération de son idPers
3. Itération sur la table Phone.
4. Récupération de 2 tuples correspondant à l'idPers
5. Accès aux 2 tuples pour en ressortir les données demandées.

Total : 6 accès à la BD.

```
{  
    "_id": ID_PERS,  
    "lastname": "Doe",  
    "firstname": "Jane",  
    "age": 33,  
    "email": "jane.doe@contoso.com",  
    "phone": [ // Rappelons que les champs peuvent être multivalués.  
        { // Un premier document  
            "type": "home",  
            "number": "555-505-4341",  
        },  
        { // Un second document  
            "type": "cellphone",  
            "number": "555-505-6864",  
        }  
    ]  
}
```

Ici, il n'y a que 2 accès à la base de données.

On peut également avoir des clients possédant plusieurs adresses e-mails (une adresse principale, une autre de récupération, etc.), ou bien ayant renseigné des informations supplémentaires comme une, ou des adresses postales.

De part sa simplicité, le modèle JSON permet une indexation à tous les niveaux, et s'adapte à l'usage que l'on en fait, et non pas l'inverse.

Référencement

Il est tout à fait envisageable de séparer des documents si le besoin s'en fait ressentir. Dans le cas d'un journal par exemple, il est normal d'avoir une collection pour les auteurs et une autre pour les articles ; à partir d'un auteur, récupérer la liste des articles qu'il a écrits, et à partir d'un article, récupérer le ou les auteurs de celui-ci. À la manière de clés primaires et étrangères, il est possible de référencer des documents grâce à leur identifiant ("`_id`").

```
{  
  "_id": "auteur_01",  
  "nom": "Doe",  
  "prenom": "John",  
  "articles": [  
    "article_01",  
    "article_04",  
    ...  
    "article_n"  
  ]  
}  
  
{  
  "_id": "article_01",  
  "titre": "Mongo c'est beau",  
  "auteurs": [  
    "auteur_01",  
    ...  
  ],  
  "date": new Date("<YYYY-mm-dd>"),  
  "commentaires": [  
    "com_01",  
    ...,  
    "com_n"  
  ]  
}
```

Bucketing

Le bucketing est une approche répandue de modélisation de données visant à séquencer efficacement des données. Il s'agit de déterminer précisément comment seront structurés chacun des documents, et sur quelle période. Par exemple, on peut imaginer des relevés sur la progression quotidienne du Covid-19 dans le monde. Un document pourrait contenir une liste de 30 relevés, et donc représenter un mois.

Une telle structure permettrait d'effectuer des calculs et prévisions avec différentes méthodes d'agrégation.

Chapitre 0 - Installation

MongoDB

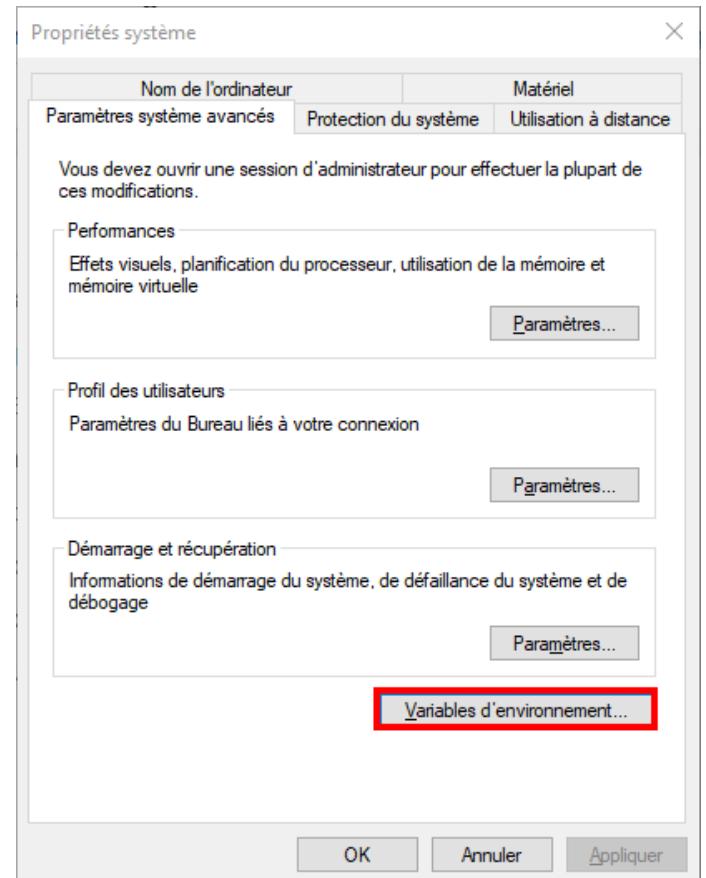
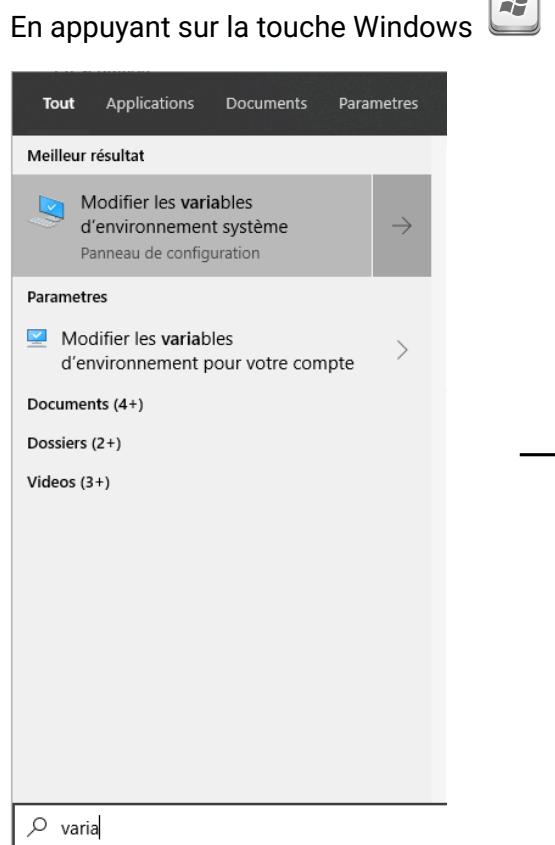
MongoDB est compatible avec toutes les familles de systèmes d'exploitations, en 64 bits. Le téléchargement se fait ici : [Download Center: Community Server](#).

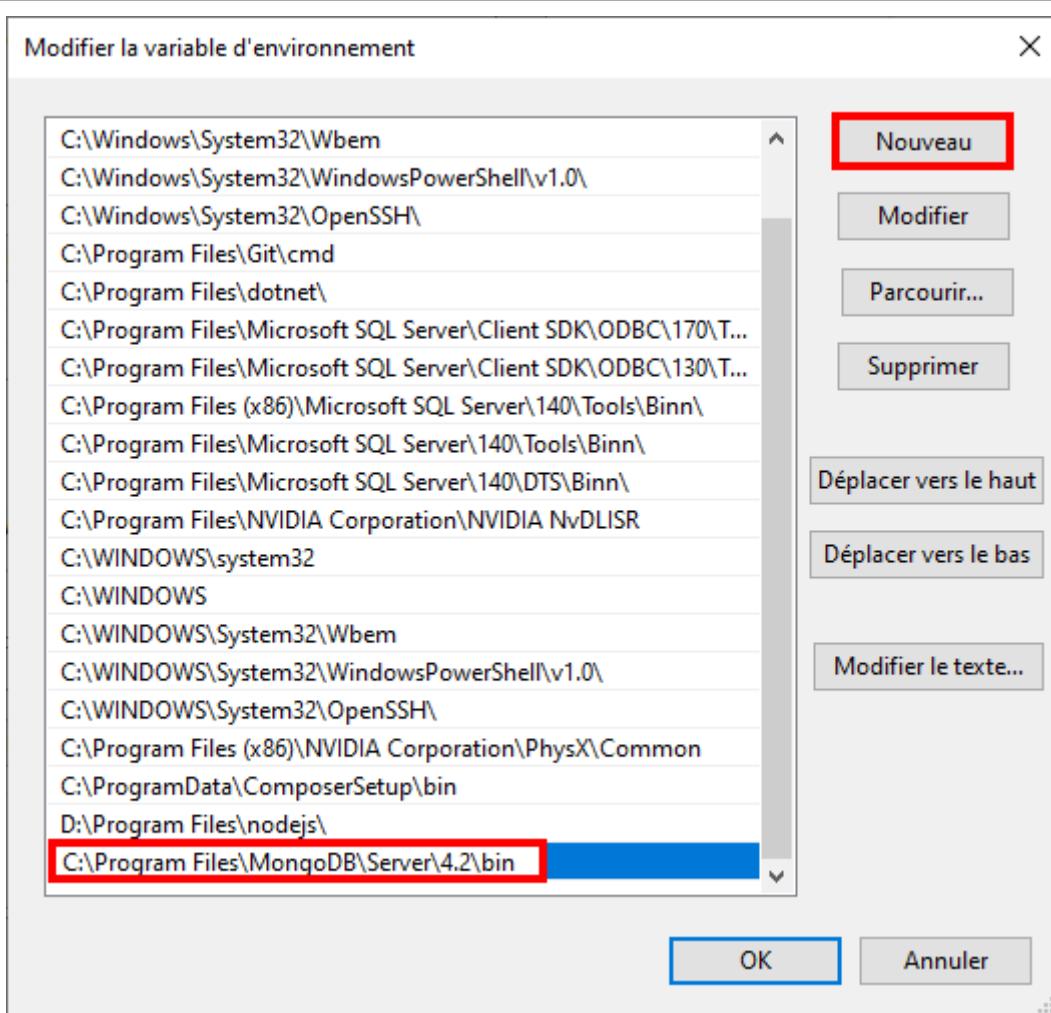
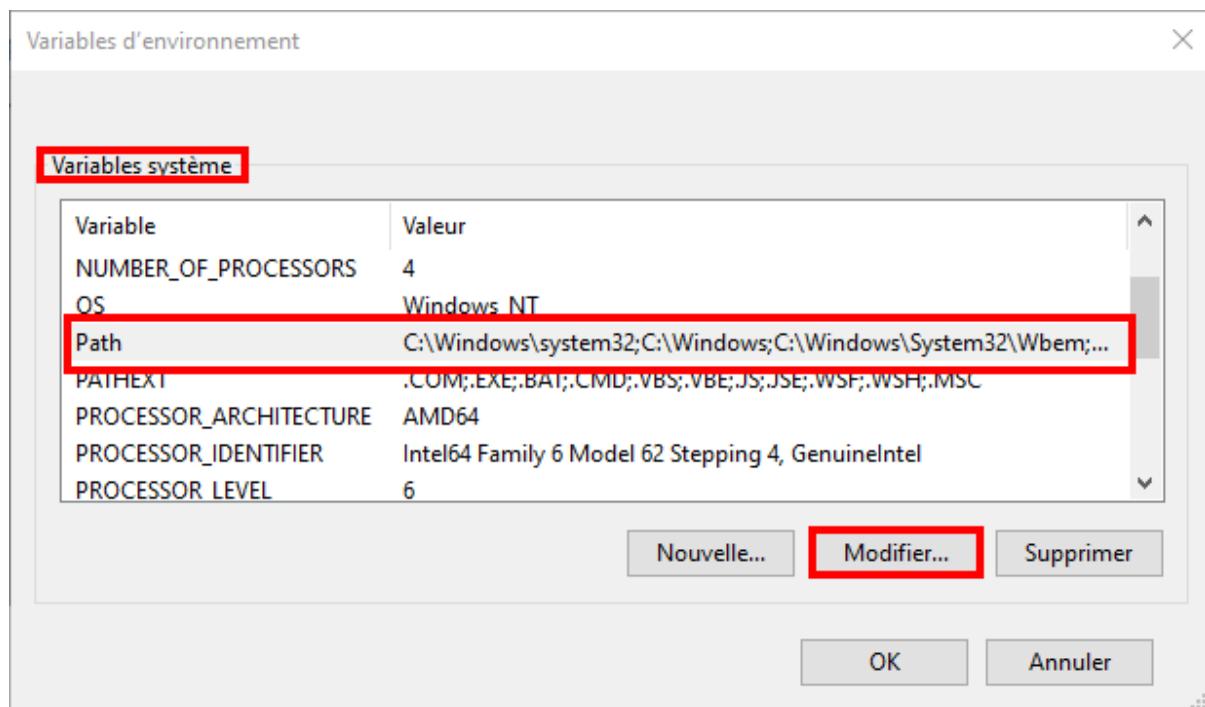
Pour profiter d'une offre gratuite, il faut sélectionner l'onglet "**Community Server**". Une fois le téléchargement terminé, lors du processus d'installation, il est recommandé de **NE PAS installer Compass** depuis cet installateur, cela aurait pour effet de télécharger la Community Edition, plus limitée.

Ajouter Mongo au PATH

Il est nécessaire d'ajouter l'invite de commande Mongo Shell à la variable d'environnement **PATH** du système afin de pouvoir l'exécuter depuis un terminal.

Windows





Note : Sous Windows, le chemin par défaut de la commande est :

C:\Program Files\MongoDB\Server\<version>\bin

Linux (Avec le gestionnaire apt)

Dans un terminal, exécuter les commandes suivantes :

Importation de la clé de chiffrement MongoDB

```
wget -qO - https://www.mongodb.org/static/pgp/server-4.2.asc | sudo apt-key add -
```

Réponse attendue: **OK**

Sinon, GPG (Gnu Privacy Guard) doit être installé avant.

```
sudo apt-get install gnupg
```

On précise la source des dépendances de MongoDB

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.2.list
```

Puis on les installe

```
sudo apt-get update
```

⚠ Ne pas confondre le package `mongodb` (maintenu par Ubuntu) et `mongodb-org` (officiel, maintenu par MongoDB). ⚠

```
sudo apt-get install -y mongodb-org
```

```
echo "mongodb-org hold" | sudo dpkg --set-selections
echo "mongodb-org-server hold" | sudo dpkg --set-selections
echo "mongodb-org-shell hold" | sudo dpkg --set-selections
echo "mongodb-org-mongos hold" | sudo dpkg --set-selections
echo "mongodb-org-tools hold" | sudo dpkg --set-selections
```

Si tout va bien, les dossiers `/var/log/mongodb/` et `/var/lib/mongodb/` ont été créés.

Pour exécuter MongoDB sur le système, il faut activer son daemon principal.

```
sudo systemctl start mongod
sudo systemctl enable mongod # permet d'activer le daemon automatiquement lors du démarrage du système
sudo status mongod # affiche l'état du daemon
sudo systemctl stop mongod # pour l'arrêter
sudo systemctl restart mongod # pour le redémarrer
```

Pour accéder à MongoDB depuis le terminal, il suffit de taper la commande `mongo`.

Compass

Compass est un **GUI (Graphical User Interface)** qui permet de **gérer les clusters Atlas**. Il représente l'équivalent de phpMyAdmin pour les bases de données MySQL, ou PhpPgAdmin pour les bases PostgreSQL.

Pour télécharger et installer Compass, rendez-vous ici : [Download Center: Compass](#).

Choisissez bien la version "Stable". L'installation devrait se dérouler sans problème, Compass est **prêt** pour l'utilisation !

Sur Linux, l'installateur est un fichier .deb exécutable, il suffit de cliquer sur *Installer*. Compass est exécutable depuis la liste d'applications, ou directement depuis le terminal avec la commande :

```
mongodb-compass
```

Présentation de l'interface :

The screenshot shows the MongoDB Compass application window. On the left, there's a sidebar with several sections highlighted by yellow boxes:

- Liste des serveurs**: Shows the cluster configuration with three hosts: cluster0-shard-00-01, cluster0-shard-00-02, and cluster0-shard-00-00.
- Description du cluster**: Shows the cluster type as Replica Set (Cluster0-shar...) and the number of nodes as 3 Nodes.
- Description de la licence**: Shows the edition as MongoDB 4.2.3 Enterprise.
- BD & Collections**: A section under the sidebar with a list of databases: admin, config, iut, local, myFirstDB, and video. Each database entry includes storage size, collection count, index count, and a delete icon.
- Creation de BD**: A button at the bottom of the sidebar.

The main area of the window is titled "Databases" and shows the same list of databases with their details. There's also a "Performance" tab and a "CREATE DATABASE" button at the top right of the main panel.

Total de documents dans la collection

DOCUMENTS 1.6k

TOTAL SIZE 341.6KB **AVG. SIZE 224B**

INDEXES 1 **TOTAL SIZE 24.0KB** **AVG. SIZE 24.0KB**

Vues : Liste JSON, Table

Document avec champs multi valued

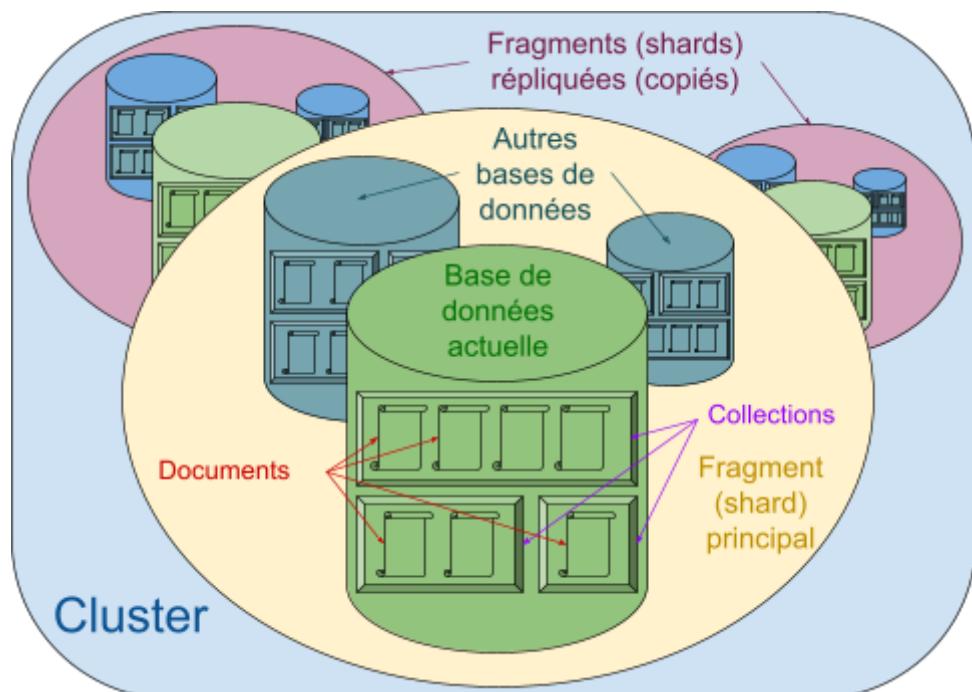
Regroupement de données représentant le type "geospatial data" de Mongo

Total de documents filtrés

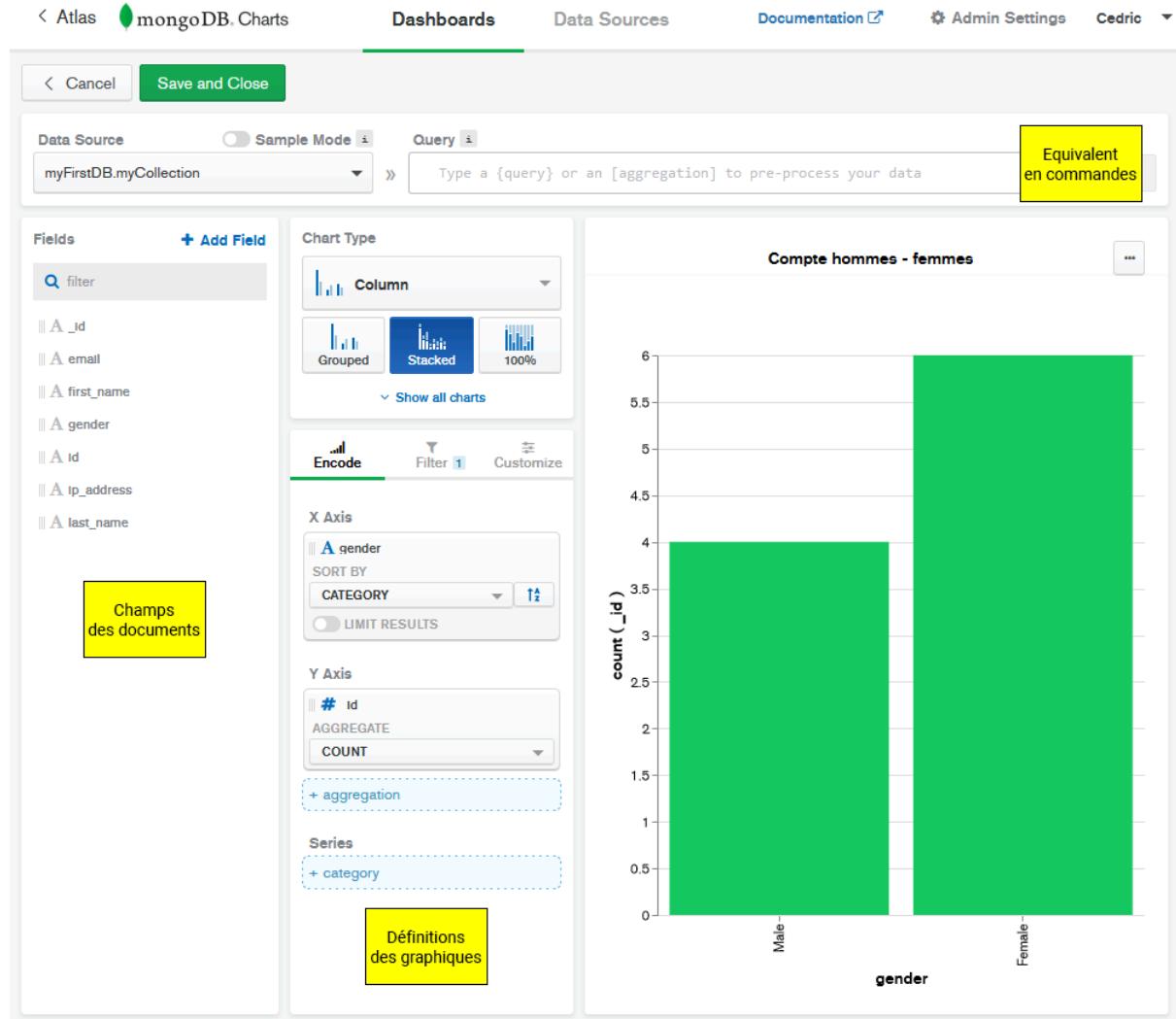
MongoDB Atlas

Présentation

MongoDB Atlas est un service cloud de type **DBaaS (DataBase as a Service)** qui simplifie l'utilisation de MongoDB via un déploiement automatique de **clusters** : il crée des **copies des données** (*replicates*) sur différents serveurs distants pour s'assurer de ne pas les perdre, et garantir leur accès même en cas d'arrêt du serveur principal.



MongoDB Atlas est une **plateforme web** accessible automatiquement après une inscription sur [le site officiel de MongoDB](#). Avec l'outil MongoDB Charts, elle permet l'étude avancée et la représentation graphique de données. Le module Stitch permet de gérer les droits d'accès des clients d'une application utilisant MongoDB.



Création d'un cluster

Après avoir créé un compte sur [Atlas MongoDB](#), différentes offres sont disponibles. L'offre "Sandbox" propose un cluster gratuit de 3 serveurs et permet un usage simple de type "proof of concept". Il faut choisir un service cloud parmi Azure, AWS et Google Cloud, chacun étant présent dans certains pays (le mieux est de choisir le plus proche). Il est possible de paramétriser les modalités de connexions dans l'onglet "Network Access", et créer des utilisateurs dans l'onglet "Database Access".

Connexion au cluster avec Compass

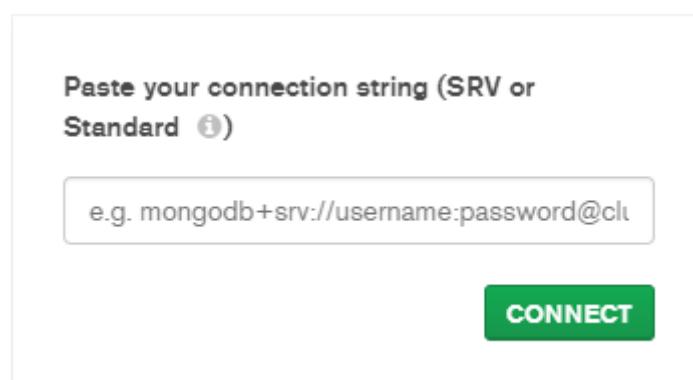
Pour se connecter à un cluster via Compass, il faut lui fournir une chaîne de caractères prédefinie. Elle est disponible à tout moment sur le panel d'Atlas, depuis la page principale du cluster.

`mongodb+srv://<username>:<password>@<cluster>.mongodb.net/test`

The screenshot shows the MongoDB Atlas Cluster Overview page for 'Cluster0'. It displays cluster details like version (4.2.3), tier (M0 Sandbox), region (AWS / Frankfurt), type (Replica Set - 3 nodes), and linked Stitch app status. A red box highlights the 'CONNECT' button. To the right, a modal window titled 'Choose a connection method' has a green box around the 'I have MongoDB Compass' option. Below the main page, a numbered list provides instructions:

- 1 Choose your version of Compass.
A dropdown menu shows '1.12 or later'.
Text: See your Compass version in "About Compass"
- 2 Copy the connection string, then open MongoDB Compass.
A code editor-like field contains a connection string:
`mongodb+srv://<REDACTED>@cluster0-tinytel.mongodb.net/test`
A 'Copy' button with a hand cursor icon is shown above the field.
Text: MongoDB Compass will auto-detect the connection string you copied. To connect, enter your database username and password into the corresponding fields when prompted.
Text: When entering your password, make sure that any special characters are [URL encoded](#).

Dans Compass, coller la chaîne obtenue en renseignant le mot de passe du compte utilisateur créé précédemment.



Connexion avec Mongo Shell

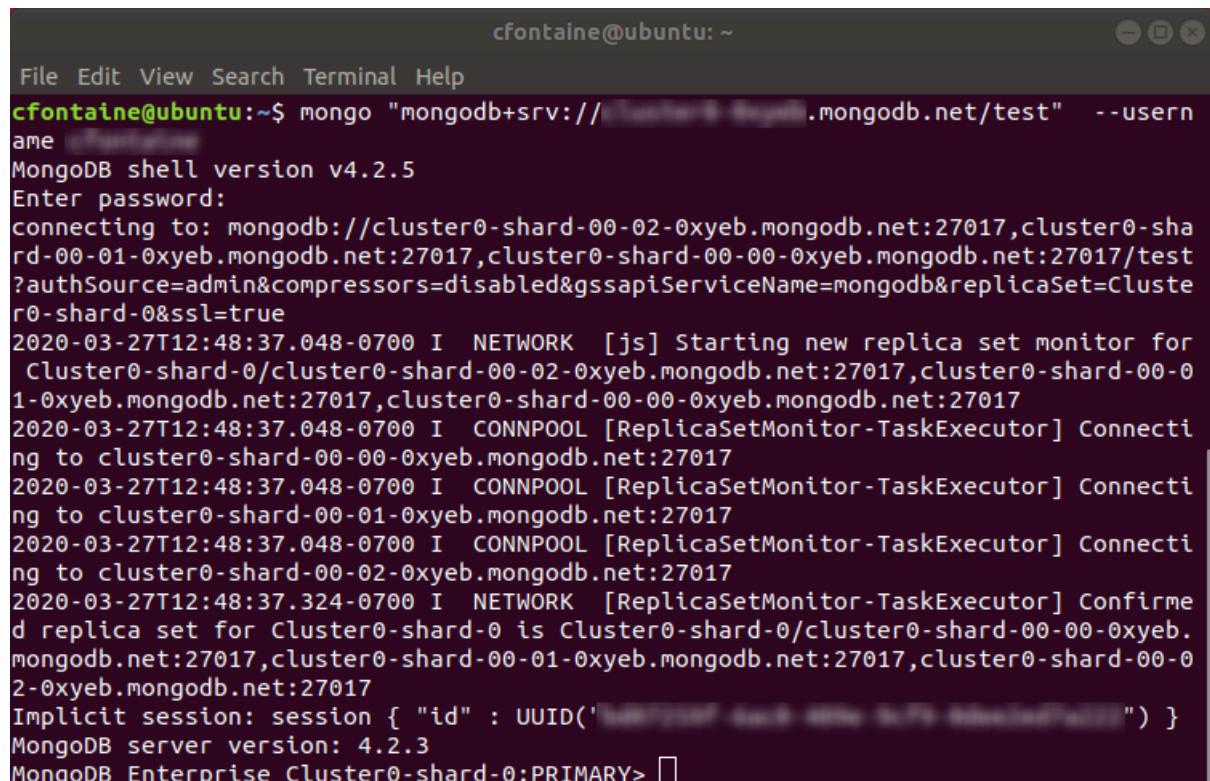
Pour réaliser différentes opérations telles que tester des commandes, exécuter des scripts, ou simplement naviguer dans la base, il est nécessaire d'avoir un accès au cluster via un terminal de commandes. Pour cela, il faut réitérer l'étape précédente en sélectionnant cette fois-ci Mongo Shell, puis entrer la commande indiquée dans le terminal.

```
mongo "mongodb+srv://<adresse_du_cluster>.mongodb.net/<BD>" --username <utilisateur> --password <mot_de_passe>
```

Note : Pour des raisons de sécurité, les options `--username` et `--password` sont facultatives. En les retirant, le terminal demandera à l'utilisateur de saisir ses informations de manière plus discrète.

Ce qui affichera le prompt suivant :

```
MongoDB Enterprise <cluster>-shard-0:PRIMARY>
```



The screenshot shows a terminal window titled 'cfontaine@ubuntu: ~'. The window contains the following text:

```
File Edit View Search Terminal Help
cfontaine@ubuntu:~$ mongo "mongodb+srv://<cluster>.mongodb.net/test" --username
MongoDB shell version v4.2.5
Enter password:
connecting to: mongodb://cluster0-shard-00-02-0xyeb.mongodb.net:27017,cluster0-shard-00-01-0xyeb.mongodb.net:27017,cluster0-shard-00-00-0xyeb.mongodb.net:27017/test
?authSource=admin&compressors=disabled&gssapiServiceName=mongodb&replicaSet=Cluster0-shard-0&ssl=true
2020-03-27T12:48:37.048-0700 I NETWORK [js] Starting new replica set monitor for Cluster0-shard-0/cluster0-shard-00-02-0xyeb.mongodb.net:27017,cluster0-shard-00-01-0xyeb.mongodb.net:27017,cluster0-shard-00-00-0xyeb.mongodb.net:27017
2020-03-27T12:48:37.048-0700 I CONNPOOL [ReplicaSetMonitor-TaskExecutor] Connecting to cluster0-shard-00-00-0xyeb.mongodb.net:27017
2020-03-27T12:48:37.048-0700 I CONNPOOL [ReplicaSetMonitor-TaskExecutor] Connecting to cluster0-shard-00-01-0xyeb.mongodb.net:27017
2020-03-27T12:48:37.048-0700 I CONNPOOL [ReplicaSetMonitor-TaskExecutor] Connecting to cluster0-shard-00-02-0xyeb.mongodb.net:27017
2020-03-27T12:48:37.324-0700 I NETWORK [ReplicaSetMonitor-TaskExecutor] Confirmed replica set for Cluster0-shard-0 is Cluster0-shard-0/cluster0-shard-00-00-0xyeb.mongodb.net:27017,cluster0-shard-00-01-0xyeb.mongodb.net:27017,cluster0-shard-00-02-0xyeb.mongodb.net:27017
Implicit session: session { "id" : UUID("XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX") }
MongoDB server version: 4.2.3
MongoDB Enterprise Cluster0-shard-0:PRIMARY>
```

Chapitre 1 - Introduction

Le principe champ-valeur

```
{  
  "champSimple": valeur1,  
  "parent": {  
    "enfant": valeur2  
  },  
  "tab": [  
    valeurT1,  
    {  
      "elem": valeurT2  
    },  
    valeurT3  
  ]  
}
```

Pour toucher à la valeur d'un champ simple : { "champSimple": valeur1 }

Pour accéder à un champ "enfant" d'un document intégré à un autre dans le champ "parent", on utilise la notation suivante : { "parent.enfant": valeur2 }

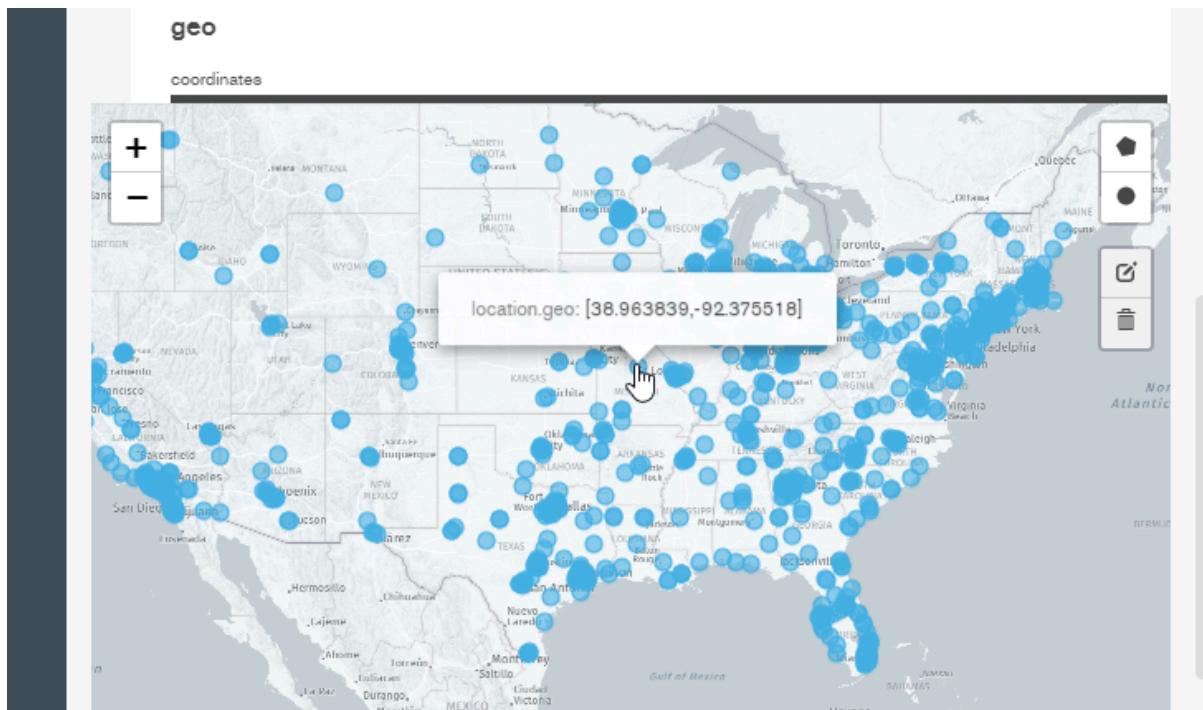
Concernant les tableaux, il y a deux méthodes :

- La recherche par élément - Pour agir sur l'élément "elem" du tableau, sans se soucier de sa position : { "tab.elem": valeurT2 }
- La recherche par position - Pour agir sur la valeur à l'indice index du tableau, quelle qu'elle soit : { "tab.index": valeurT3 } (ex : { "tab.0": valeurT1 })

Exemple de type composite : Les données géospatiales

MongoDB supporte des types composites prenant la forme d'un document. Par exemple, le type geospatial data, qui est composé d'une chaîne de caractère (type) et d'un tableau de N nombre décimaux (coordinates). Cet objet peut ainsi être représenté comme un point dans un repère de dimension N. Ainsi, on obtient un point en deux dimensions aux coordonnées (x, y) = (-7, 3.8) :

```
{  
  "type": "Point",  
  "coordinates": [  
    -7,  
    3.8  
  ]  
}
```



Comprendre l'analyse de schémas sur Compass

Documents Aggregations **Schema** Explain Plan Indexes Validation

Compass propose un outil permettant d'analyser les schémas de la base de données. Cet outil fournit des détails sur tous les types (composites et natifs) utilisés dans les documents, et des statistiques sur leur utilisation.

On peut connaître la part que représente une valeur sur un ensemble de valeurs, visualiser des coordonnées, et avoir une vue d'ensemble sur la totalité des occurrences, d'un simple clic.

Cela apporte également une aide nécessaire à la conception du schéma de la base de données, en raison de la flexibilité du format BSON.

Exemple :

Soit une collection composée d'étudiants dans un IUT. On peut connaître les moyennes de chaque étudiants, et le nombre d'élèves par semestre.

iut.etudiant Schema

iut.etudiant

DOCUMENTS 38 TOTAL SIZE 22.8KB AVG. SIZE 613B INDEXES 1 TOTAL SIZE 36.0KB AVG. SIZE 36.0KB

Documents Aggregations Schema Explain Plan Indexes

FILTER OPTIONS ANALYZE RESET ...

Query returned 38 documents. This report is based on a sample of 38 documents (100.00%). ⓘ

_id

objectid

ObjectId est un type natif
100 % des documents en possèdent un

S M T W T F S 0:00 6:00 12:00 18:00 23:00

first: 2020-03-27 12:26:30 last: 2020-03-28 21:58:31

classe

string

97 % des documents possèdent un champ "classe" de type string
Les 3 % restants sont undefined (n'existent pas)

S1 32 %

S1 S3 S2 S4

Liste des valeurs rencontrées dans le champs "classe"
Compass offre des statistiques, ce qui permet de connaître la part des étudiants dans chaque classe

nom

string

100% des documents possèdent un champ "nom" de type string

5%
2,5%
0%

Le pourcentage d'apparition de chaque valeur de "nom"
Ici, 5% des étudiants ont le même nom

notes

array

document

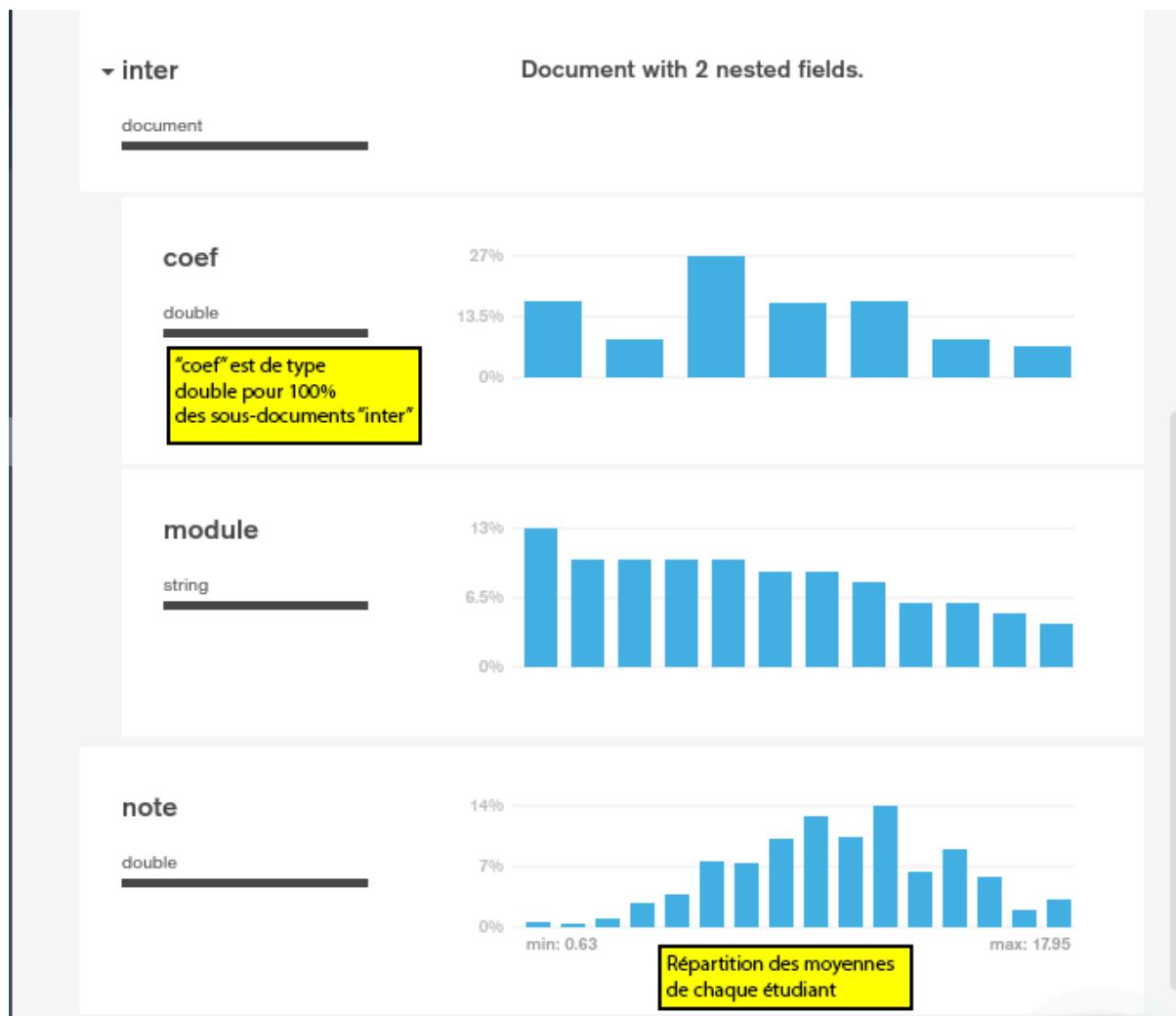
"notes" est pour 97% des étudiants un tableau de documents
N'existe pas pour les 3 % restant (undefined)

min: 1 average: 8.24 max: 10

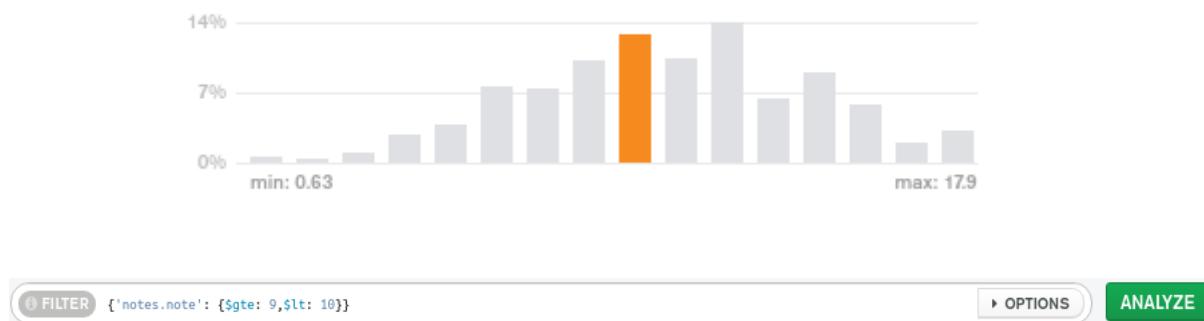
Array of documents with 2 nested fields.
Array lengths
Description du contenu de "notes": Des documents composés de 2 champs
Détails sur la taille de chaque tableau de notes

Velazquez Burks Langley Klein Glenn
McKinney Best Donovan Rogers Joyce
Kennedy Santos Guy Lawrence John Foster





En cliquant sur l'une des barres dans l'un des graphiques, cela crée une requête dans la zone de filtre, permettant de voir tous les étudiants dont la moyenne correspond à l'intervalle sélectionné :



Il est donc possible de parcourir à volonté tout le dataset et d'en faire une synthèse lisible.

Chapitre 2 - CRUD

- Pré-requis: [iut.js - ownCould IUT Montreuil](#) (Disponible sur Moodle)

Préparation

L'acronyme “**CRUD**” signifie :

- **C**reate (Création / Insertion)
- **R**ead (Lecture / Agrégation)
- **U**pdate (Mise à jour)
- **D**elete (Suppression)

Il désigne donc l'ensemble des opérations réalisables sur une base de données. Le Mongo Shell est un environnement d'exécution de JavaScript. Il est possible d'y charger des scripts écrits en JS permettant la création, et la manipulation de bases de données. On charge un script avec la fonction `load()` :

```
load("iut.js");
```

Si le script est chargé, la réponse `true` est affichée dans le terminal.

Le chemin du fichier est relatif au répertoire courant (là où a été exécuté la commande `mongo`). Le Shell propose de nombreuses commandes natives, permettant entre autres de naviguer dans les répertoires :

```
quit(); // permet de quitter l'environnement Mongo
exit // idem
cd(path); // reçoit en paramètre une chaîne de caractères indiquant le
// chemin d'un répertoire
pwd(); // affiche le répertoire courant
ls(); // affiche le contenu du répertoire courant
cat(file); // affiche le contenu d'un fichier
mkdir(dir); // crée un sous-répertoire dans le répertoire courant
```

Read

Depuis Mongo Shell, il est possible d'effectuer des requêtes (*queries*) pour récupérer des documents. Cependant, il faut préciser où chercher ces documents (`dataBase.collection`). Pour lister les bases de données disponibles :

```
show dbs
```

Si le script `iut.js` a été chargé, la commande ci-dessus affiche :

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> show dbs
admin      0.000GB
iut        0.000GB
local     3.842GB
```

Équivalent PSQL :

```
postgres=# \dn
```

Il faut ensuite sélectionner la base de donnée :

```
use iut
MongoDB Enterprise Cluster0-shard-0:PRIMARY> use iut
switched to db iut
```

Équivalent PSQL :

```
postgres=# SET search_path TO iut;
```

Une fois dans la base, on peut lister les collections avec :

```
show collections
```

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> show collections
departement
enseignant
etudiant
module
personnel
```

Équivalent PSQL :

```
postgres=# \dt
```

Lorsqu'une base de données est sélectionnée, MongoDB définit une variable globale "db" qui sert de référence vers celle-ci. C'est un objet possédant divers attributs et méthodes, notamment sa liste de collections, qui sont elles-mêmes des objets :

```
db.collectionName
```

Les collections possèdent des méthodes pour traiter des données, dont `.find()` qui permet de récupérer des documents. C'est cette méthode qui récupère les filtres de requête et de projection. Elle peut prendre jusqu'à 2 arguments et renvoie un curseur (qui n'est ni plus ni moins qu'une sous-collection) contenant les documents filtrés. Ce curseur est donc également un objet avec différentes méthodes permettant de manipuler les documents.

Mise en application du BSON

Les documents "plats"

Ces documents sont les plus proches des tuples dans un SGBDR. Commençons par récupérer la liste des enseignants :

```
db.enseignant.find();
```

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.enseignant.find()
{
  "_id" : 5, "nom" : "Rety", "prenom" : "Jean-Hugues", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 7, "nom" : "Lamolle", "prenom" : "Myriam", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 8, "nom" : "Homps", "prenom" : "Marc", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 9, "nom" : "Simonet", "prenom" : "Marianne", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 10, "nom" : "Ricordeau", "prenom" : "Anne", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 11, "nom" : "Georges", "prenom" : "Rémi", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 12, "nom" : "Delmas", "prenom" : "Guylain", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 13, "nom" : "Nauwynck", "prenom" : "Nedra", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 15, "nom" : "Bonnot", "prenom" : "Philippe", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 16, "nom" : "Clément-Comparot", "prenom" : "Véronique", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 17, "nom" : "Le Duc", "prenom" : "Chan", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 18, "nom" : "Bossard", "prenom" : "Aurélien", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 19, "nom" : "Cataldi", "prenom" : "Mario", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 20, "nom" : "Golven", "prenom" : "Amélie", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 21, "nom" : "Ballay-Dally", "prenom" : "Charlotte", "titulaire" : true, "dept" : "QLIO" }
{
  "_id" : 23, "nom" : "Groff", "prenom" : "Geoffrey", "titulaire" : true, "dept" : "INFO" }
{
  "_id" : 22, "nom" : "Chebbi", "prenom" : "Imen", "titulaire" : false }
{
  "_id" : 24, "nom" : "Tobbelem", "prenom" : "Jocelin", "titulaire" : false }
{
  "_id" : 25, "nom" : "Mocket", "prenom" : "Mehdi", "titulaire" : false }
{
  "_id" : 26, "nom" : "Mourel", "prenom" : "Frédéric", "titulaire" : false }
}
Type "it" for more
```

NB : Le curseur n'affiche que 20 documents. Pour afficher la suite, il suffit de taper "it" (*iterate*). À contrario, ajouter `.limit(n)` à la requête limitera le résultat à `n` documents.
Par ailleurs, la méthode `.pretty()` permet d'embellir l'affichage des documents.

Équivalent SQL :

```
SELECT * FROM iut.enseignant;
```

Le premier argument de la méthode `.find()` permet de spécifier un filtre. C'est donc un objet "modèle type" des documents souhaités. Petit exemple pour filtrer les enseignants du département informatique :

```
db.enseignant.find({ "dept" : "INFO" }).pretty();
```

Équivalent SQL :

```
SELECT * FROM iut.enseignant WHERE dept = 'INFO';
```

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.enseignant.find({ "dept" : "INFO" }).pretty()
{
  "_id" : 5,
  "nom" : "Rety",
  "prenom" : "Jean-Hugues",
  "titulaire" : true,
  "dept" : "INFO"
}
{
  "_id" : 7,
  "nom" : "Lamolle",
  "prenom" : "Myriam",
  "titulaire" : true,
  "dept" : "INFO"
}
```

Les enseignant dont le champ "dept" était différent de "INFO" ou non renseigné (notion d'`undefined`) n'ont pas été retournés par la requête. Le champ "`_id`" est un champ spécial de MongoDB qui doit être unique au sein d'une collection (équivalent de clé primaire d'une table) et est obligatoirement présent dans tous les documents. Nous pouvons ainsi récupérer un enseignant depuis son identifiant pour être sûr de n'en récupérer qu'un seul :

```
db.enseignant.find({ "_id": 7 }).pretty();
```

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.enseignant.find({ "_id": 7 }).pretty()
{
    "_id" : 7,
    "nom" : "Lamolle",
    "prenom" : "Myriam",
    "titulaire" : true,
    "dept" : "INFO"
}
MongoDB Enterprise Cluster0-shard-0:PRIMARY>
```

Les documents "intégrés"

Pour appliquer la notion de champs multi valués, possible par le format JSON, la collection des étudiants est un bon exemple. Chaque étudiant peut avoir une liste de notes, et chaque note peut également être un document spécifiant sa valeur et le module correspondant.

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.etudiant.find().pretty()
{
    "_id" : ObjectId("5e7df0f692a05774eed7d74f"),
    "nom" : "Charles",
    "prenom" : "Glenn",
    "classe" : "S3",
    "notes" : [
        {
            "inter" : {
                "module" : "M3202c",
                "coef" : 2.5
            },
            "note" : 9.24
        },
        {
            "inter" : {
                "module" : "M2202",
                "coef" : 1.5
            },
            "note" : 11.16
        }
    ]
}
```

Projection

La projection permet de spécifier uniquement les champs des documents filtrés. Elle s'effectue en passant un deuxième document dans la requête, précisant quels champs seront affichés ou ignorés, avec les valeurs 1 et 0.

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.etudiant.find({}, { "_id": 0, "nom": 1, "prenom": 1 })
{
    "nom" : "Charles", "prenom" : "Glenn" ,
    "nom" : "Barton", "prenom" : "Sellers" ,
    "nom" : "Marion", "prenom" : "Contreras" ,
    "nom" : "Pearson", "prenom" : "Santos" ,
    "nom" : "Bradshaw", "prenom" : "Kennedy" ,
    "nom" : "Carissa", "prenom" : "Garcia" ,
    "nom" : "Rosalind", "prenom" : "Lawrence" ,
    "nom" : "Finch", "prenom" : "Wooten" ,
    "nom" : "Trudy", "prenom" : "Pace" ,
    "nom" : "Vinson", "prenom" : "Ayers" ,
    "nom" : "Keith", "prenom" : "Booker" ,
    "nom" : "Petty", "prenom" : "Brewer" ,
    "nom" : "Lowery", "prenom" : "Gordon" ,
    "nom" : "Kayla", "prenom" : "Rogers" ,
    "nom" : "Jessica", "prenom" : "Joyce" ,
    "nom" : "Burt", "prenom" : "Foster" ,
    "nom" : "Chen", "prenom" : "Reilly" ,
    "nom" : "Mclean", "prenom" : "Ortega" ,
    "nom" : "Whitney", "prenom" : "Best" ,
}
```

Équivalent SQL :

```
SELECT nom, prenom FROM iut.etudiant;
```

Quels sont les noms des étudiants en S3 ?

```
db.etudiant.find({"classe": "S3"}, {"_id": 0, "nom": 1, "prenom": 1});
```

Create

Il est possible d'ajouter des documents depuis Compass.

The screenshot shows the MongoDB Compass interface. On the left, there's a sidebar with a tree view labeled 'iut.etudiant'. Under 'Documents', there are three documents listed with their IDs and some fields. In the center, a modal window titled 'Insert to Collection iut.etudiant' is open. It contains a code editor with the following JSON:

```
1 ↴ []
2 ↴ [
3   {
4     "nom": "Doe",
5     "prenom": "John",
6     "classe": "S3"
7   },
8   {
9     "nom": "Doe",
10    "prenom": "Jane"
11  }
12 ]
```

On the right side of the modal, the results of the insertion are shown in a list:

- _id: ObjectId("5e7df0f692a05774eed7d772")
nom: "Barbra"
prenom: "Spencer"
classe: "S1"
> notes: Array
- _id: ObjectId("5e7fc88758390c5d38076917")
nom: "Doe"
prenom: "John"
classe: "S3"
- _id: ObjectId("5e7fc88758390c5d38076918")
nom: "Doe"
prenom: "Jane"

At the bottom of the modal, there are 'CANCEL' and 'INSERT' buttons. The 'Insert Document' button is highlighted with a mouse cursor.

Et depuis le Mongo Shell, via 2 fonctions :

```
db.etudiant.insertOne({
  "nom": "Doe",
  "prenom": "John"
});

db.etudiant.insertMany([
  {
    "nom": "Doe",
    "prenom": "John",
    "classe": "S3"
  },
  { // Note : Les notes et la classe n'ont pas été renseignées
    "nom": "Doe",
    "prenom": "Jane"
  }
]);
```

Équivalent SQL :

```
INSERT INTO iut.etudiant (nom, prenom, classe) VALUES ('Doe', 'John', 'S3');
INSERT INTO iut.etudiant (nom, prenom) VALUES ('Doe', 'Jane');
```

Update

Pour faire passer John Doe en S4 et lui ajouter des notes depuis Compass, il suffit de cliquer sur le bouton d'édition du document.

```
_id: ObjectId("5e7fc88758390c5d38076917")
nom: "Doe"
prenom: "John"
classe: "S3"
```

En mode édition, il est possible de :

- Modifier les noms des champs, ainsi que leur valeur, en cliquant dessus.
- Ajouter **+** et supprimer **x** des champs

```
1 _id: ObjectId("5e7fc88758390c5d38076917") ObjectId
2 nom : "Doe" String
3 prenom : "John " String
4 classe : "S4 " String▼
5 + Add Field After classe
Document Modified.
```

CANCEL UPDATE

- Changer le type des données

```
notes : " "
String▼
Array
```

```
1 _id: ObjectId("5e7fc88758390c5d38076917") ObjectId
2 nom : "Doe" String
3 prenom : "John " String
4 classe : "S4 " String
5 notes : Array
6   0 : Object
7     inter : Object
8       module : "M2106 " String
9       coef : 1.5 Double
10      note : 17.83 Double
```

CANCEL UPDATE

L'équivalent avec Mongo Shell s'effectue avec les fonctions `updateOne()` et `updateMany()`, qui prennent en arguments un filtre, et un document contenant l'opérateur de mise à jour.

```
db.etudiant.[updateOne | updateMany](<filtre>, <update>, <arguments additionnels>);
```

- <filtre> : Document modèle permettant de récupérer les objets à modifier.
- <update> : Un document contenant un opérateur, soit l'action à effectuer (éditer, supprimer, incrémenter, etc.). Cet opérateur prend également pour valeur un document composé des champs à modifier, et leur(s) nouvelle(s) valeur(s).
- <arguments additionnels> : permet d'ajouter un filtre en cas de champs multi-valués (liste de valeurs) pour ne modifier que les sous-valeurs répondant à ce filtre.

Les différents opérateurs

Opérateur	Description	Exemple
Manipulation de champs		
\$set	Affecte une valeur à un champ	db.collection.update({}, {\$set: {champ: valeur}});
\$unset	Supprime un champ	db.collection.update({}, {\$unset: {champ: " "}});
\$rename	Renomme un champ	db.collection.update({}, {\$rename: {champ: "nouveauNom"}});
\$mul	Multiplie la valeur d'un champ	db.collection.update({}, {\$mul: {champ: 5}});
\$inc	Incrémente la valeur d'un champ	db.collection.update({}, {\$inc: {champ: 2}});
\$min	Modifie la valeur d'un champ si la nouvelle valeur est inférieure à la valeur actuelle	db.collection.update({}, {\$min: {champ: 0}}); <i>// Le champ prendra la valeur 0 si la valeur actuelle est plus grande</i>
\$max	Modifie la valeur d'un champ si la nouvelle valeur est supérieure à la valeur actuelle	db.collection.update({}, {\$max: {champ: 100}}); <i>// Le champ prendra la valeur 100 si la valeur actuelle est plus petite</i>
Manipulation de tableaux		
\$	Modifie le premier élément d'un tableau.	db.collection.update({champ: valeur}, {"champ.\$": valeur});
\$[]	Modifie tous les éléments d'un tableau	db.collection.update({champ: valeur}, {"champ.\$[]": valeur});

\$pop	Retire le premier élément d'un tableau	db.collection.update({}, {\$pop: {champ: -1}});
\$pull	Retire tous les éléments filtrés d'un tableau	db.collection.update({}, {\$pull: {champ: valeur}});
\$pullAll	Retire tous les éléments d'un tableau qui correspondent à une valeur donnée	db.collection.update({}, {\$pullAll: {tableau: valeur}});
\$push	Ajoute un élément au tableau	db.collection.update({}, {\$push: tableau: { \$each: [1, 2, 3, 4]}}); // \$each permet d'indiquer que l'on ajoute chacun des éléments (individuellement), non pas le tableau en tant que tel
\$addToSet	Ajoute un élément au tableau s'il n'est pas déjà présent	db.collection.update({}, {\$addToSet: {tableau: élément}});
[\$<id>]	<id> est un alias générique. Modifie tous les éléments d'un tableau selon la condition passée en arguments additionnels à la fonction update() via l'attribut natif arrayFilters	db.collection.update({}, {"tableau.\$[id]": valeur}, {arrayFilter: [{id: condition}]})); // Affecte une valeur à tous les éléments d'un tableau qui correspondent à la valeur donnée

updateOne()

La fonction updateOne() permet de mettre à jour le premier document retourné par le filtre. Par exemple, faire passer John Doe en S4 :

```
db.etudiant.updateOne({'nom': 'Doe', 'prenom': 'John'}, {$set:
{'classe': 'S4'}});
// Le premier document avec pour nom et prénom Doe, John reçoit S4 en
valeur du champ Classe
```

updateMany()

La fonction updateMany(), elle, met à jour tous les éléments filtrés.

Par exemple, on souhaite ajouter 1 point bonus à tous les étudiants qui ont passé une évaluation dans un module donné.

Contraintes :

- Tous les étudiants n'ont pas de notes dans les mêmes modules.
- Le champ "notes" est un tableau de documents. Chaque document contient une note, et un autre sous-document "inter" dans lequel est indiqué le nom du module.

```

_id: ObjectId("5e7df0f692a05774eed7d74f")
nom: "Charles"
prenom: "Glenn"
classe: "S3"
notes: Array
  0: Object
    inter: Object
      module: "M3202C"
      coef: 2.5
      note: 9.24
  1: Object
  2: Object
  3: Object
  4: Object
  5: Object
  6: Object
  7: Object

```

Le meilleur choix est donc d'utiliser l'opérateur `$[<id>]` qui permettra de filtrer les notes et incrémenter uniquement celles qui correspondent au module souhaité (e.g : "M1104").

NB : Cela implique le filtrage du tableau notes (`arrayFilters`).

L'incrémantion se fait avec `$inc` (cf : tableau des opérateurs).

Mise en application :

- Le filtre - Récupère les étudiants ayant passé l'interro pour le module donné :
`{"notes.inter.module": "<Module>"}`
- Update - Le document peut lui-même être décomposé de la sorte :
`{$<opérateur>: {"<champ>": <valeur>}}`
 - L'opérateur - Incrémenter les notes : `$inc`
 - Le champ - Les notes étant des champs multivalués, on accède à la valeur "note" via la notation suivante : `"notes.$[id].note"`.

NB : l'élément du tableau sur lequel on itère est indexé sur `$[<id>]`. Cela signifie que lors de la mise à jour des éléments, id servira de référence à chaque document compris dans le tableau de notes de l'étudiant.

- La valeur - Un point bonus, donc 1
- Les arguments additionnels - La fonction update reçoit un 3ème document en paramètre, qui a un attribut `arrayFilters`, pour appliquer une condition aux éléments indexés par l'opérateur `$[<id>]`. Cet attribut est donc un filtre supplémentaire :
`{"id.inter.module": "<Module>"}`

```

db.etudiant.updateMany({"notes.inter.module": "<Module>"},
  {$inc: {"notes.$[id].note": 1}},
  {arrayFilters: [
    {"id.inter.module": "<Module>"}
  ]
});

```

Cependant, il est possible qu'avec ce point bonus, certains étudiants aient une note supérieure à 20. Pour pallier ce problème, on pourrait filtrer les notes dépassant ce seuil et affecter la valeur 20 aux notes qui dépassent le seuil maximal. L'opérateur \$min répond à ce besoin.

```
db.etudiant.updateMany({"notes.inter.module": "<Module>"},  
    {$min: {"notes.note": 20}}  
);
```

Un équivalent en Java serait :

```
// notes représente le tableau de notes de chaque étudiant  
for (Document id : notes) { // id est la variable qui itère sur chaque élément du tableau notes  
    if (id.inter.module.equals("<Module>")) { // si le module du sous-document inter est bien M1104  
        if (++id.note > 20) { // on incrémente la valeur de l'attribut "note"  
            id.note = 20; // si elle est supérieure à 20, la mettre au max  
        }  
    }  
}
```

Ou bien en SQL :

```
CREATE OR REPLACE FUNCTION bonusInter(module varchar, bonus int) RETURNS void AS $$  
    UPDATE notes SET note = note + bonus WHERE idInter IN (SELECT idInter FROM interro WHERE idMod = module);  
    UPDATE notes SET note = 20 WHERE note > 20;  
$$ LANGUAGE 'sql';
```

Delete

Enfin, il est possible de supprimer totalement les documents de la même manière qu'ils sont créés, lus et modifiés, avec les méthodes deleteOne() et deleteMany(), qui n'ont de différent que le nom. La première retire la première occurrence correspondant au filtre, tandis que l'autre les supprime toutes.

```
db.collection.[ deleteOne | deleteMany ](<filtre>);
```

Par exemple, pour supprimer tous les étudiants dont le nom est Doe :

```
db.etudiant.deleteMany({'nom': 'Doe'});
```

Sur Compass il suffit de cliquer sur l'icône de suppression  en passant sur le document.

```
> _id: ObjectId("5e7fc88758390c5d38076917")  
  nom: "Doe"  
  prenom: "John"  
  classe: "S4"  
> notes: Array
```



Chapitre 3 - Query language

Les filtres sont des objets JSON permettant de donner des conditions sur les documents à accepter (équivalent du WHERE en SQL).

Les filtres d'égalité

Les filtres d'égalités s'opèrent en comparant les documents à un objet. Pour récupérer tous les documents de la collection Client dont le prénom est "John", on applique le filtre suivant :

```
{  
  "firstname": "John"  
}
```

La requête renvoie la liste des documents respectant la condition :

```
{ // Ici, en l'occurrence, un seul document.  
  "_id": 2,  
  "lastname": "Doe",  
  "firstname": "John",  
  "city": "Paris"  
}
```

Équivalent SQL :

```
SELECT * FROM Client WHERE firstname = 'John';
```

Les filtres de portée

Les filtres de portée permettent une évaluation plus souple des documents, comme l'appartenance à un interval pour des nombre. Pour cela il faut utiliser des opérateurs : \$gt[e] et \$lt[e].

Afin de filtrer les voitures dont le prix est compris entre 1 000 (inclus) et 2 500 (exclus), le champ "value" reçoit l'objet ayant pour propriétés les conditions à appliquer :

```
{  
  "value": {  
    "$gte": 1000,  
    "$lt": 2500  
  }  
}
```

```
{
  "model": "Peugeot 207",
  "year": 2013,
  "color": "White",
  "value": 1500.00,
  "owner": 1
}
```

```
SELECT * FROM Cars WHERE value >= 1000 AND value < 2500;
```

Rappelons que le JSON est une structure multivaluée. Il est donc possible de combiner les filtres pour exécuter des requêtes plus pointues.

Les principaux opérateurs

Les opérateurs de MongoDB sont des champs JSON particuliers. Leur nom est précédé par le symbole \$, dans le but de les différencier de tout autre champ quelconque d'un document. Ils prennent une ou plusieurs valeurs en entrée et ressortent une autre valeur, et ont une syntaxe similaire au VBA (Excel).

Unaires

Les opérateurs unaires prennent une unique valeur et renvoient un résultat :

Opérateur	Description	Exemple
\$eq	Égalité	{"\$eq": 5}
\$ne	Inégalité	{"\$ne": "Peugeot 207"}
\$not	NON logique	{"\$not": {"\$eq": "Dacia Duster"}}
\$gt[e]	Suppériorité Greater than (or equal)	{"\$gt": 7} {"\$gte": -3}
\$lt[e]	Infériorité Less than (or equal)	{"\$lt": -4} {"\$lte": 2}
\$in	Vérifie si un champ a pour valeur un élément d'une liste donnée	{"\$in": [val1, val2, ..., val_n]}
\$nin	Inverse de \$in	{"\$nin": [val1, val2, ..., val_n]}

Logique

Opérateur	Description	Exemple
\$and	ET	{"\$and": [{"\$gt": 6}, {"\$ne": 8}]}]
\$or	OU	{"\$or": [{"\$eq": -3}, {"\$eq": 1}]}]
\$not	NOT inverse le filtre	{"\$not": {"\$gt": 5}} // inférieur à 5

Evaluation

Certains opérateurs permettent "d'évaluer" des champs selon des conditions plus strictes. C'est le cas par exemple de l'opérateur \$regex, qui vérifie qu'une chaîne vérifie une expression régulière. Voici une démonstration avec une expression qui vérifie que la chaîne correspond à un nom de domaine, et l'opérateur \$options précisant les différents drapeaux (*flags*) de l'expression :

```
{
  "$regex": "^((http[s]?:\/\/)?([a-z]+\.)*[a-z]*\.[a-z]{2,})\/?$/,
  "$options": "i" // i signifiant que l'on ignore la casse
}
```

Opérateur	Description	Exemple
\$regex	Vérifie une expression régulière	{champ: {"\$regex": /exp/, "\$options": "[igmsuy]"}}
\$where	Vérifie si un champ répond à une condition passée dans une fonction en JavaScript	{champ: {"\$where": function() { return condition; }}}
\$mod	Vérifie le modulo d'un champ selon un diviseur donné	{champ: {"\$mod": [diviseur, reste]}} {champ: {"\$mod": [2, 0]}}

Validation

Il est également possible de filtrer les documents selon l'existence et le type d'un champ.

Opérateur	Description	Exemple
\$exists	Booléen : vérifie si le champ est défini ou non	{champ: {"\$exists": true false}}
\$type	Vérifie si le champ est du type donné	{champ: {"\$type": "string" "double" "int32" "..." }}

Tableaux

Opérateur	Description	Exemple
\$all	Vérifie qu'un tableau contient toutes les valeurs spécifiées dans la requête	{champ: {"\$all": [val1, val2..., val_n]}} // vérifie que toutes les val1 à val_n sont existent dans le tableau "champ"
\$size	Vérifie la longueur du tableau	{champ: {"\$size": 4}} // Vérifie que la taille du tableau est 4
\$elemMatch	Vérifie qu'au moins un élément d'un tableau correspond aux conditions spécifiées dans la requête	{champ: {"\$elemMatch": { \$gte: 5, \$lt: 10 }} // Vérifie qu'au moins un élément est compris entre 5 et 10 }}

Chapitre 4 - Procédures stockées

⚠ Fonctionnalité réservée aux clusters payants, l'offre gratuite ne supportant pas le JavaScript côté serveur ⚡

Création

Il est possible de stocker des fonctions écrites en JavaScript sur le serveur. La base de données possède une collection native nommée "system.js".

On sauvegarde ces fonctions via la méthode db.system.js.insertOne() :

```
db.system.js.insertOne({  
    "_id": "helloWorld",  
    "value": function() {  
        return "Hello World!";  
    }  
});  
  
db.system.js.insertOne({  
    "_id": "affMoyenneEtud",  
    "value": function(etud) {  
        let avg = 0;  
        // Vérifie que etud.notes != undefined et etud.notes != null  
        if (etud.notes && etud.notes.length != 0) {  
            let somCoef = etud.notes.reduce((s, v) => s + v.inter.coef, 0);  
            avg = etud.notes.reduce((a, v) => a + v.note * v.inter.coef, 0) /  
            somCoef; // Calcule la moyenne avec les coefficients  
        }  
        return etud.prenom + " " + etud.nom + " : " + avg + "/20";  
    }  
});
```

NB : Pour mettre à jour une fonction, on utilisera plutôt la méthode .replaceOne(), similaire à .updateOne() à l'exception qu'on ne peut pas utiliser les opérateurs d'update.

Utilisation

Ces procédures sont utilisables n'importe où, et peuvent également servir à filtrer des documents (dans un \$where par exemple).

Exemple d'utilisation : Afficher tous les étudiants avec leur moyenne

```
db.etudiant.find().forEach(etud => affMoyenneEtud(etud));
```

Annexes

Sources

- [Initiation à la conception de schémas](#)
- [MongoDB University - M001](#)
- [MongoDB Documentation](#)

Documentation

La documentation de MongoDB est très bien expliquée et complète, c'est le meilleur endroit pour chercher une information : [The MongoDB 4.2 Manual – MongoDB Manual](#)

La section “Références” recense toutes les méthodes, regroupées par catégories : [Reference – MongoDB Manual](#)

Bréviaire - CRUD

Create

```
db.collection.insertOne(<document>);

db.collection.insertMany([
    <document1>,
    <document2>,
    ...
]);
```

Équivalent SQL :

```
INSERT INTO db.collection (<champs_des_documents>) VALUES
(<valeurs_du_document_1>),
(<valeurs_du_document_2>),
...;
```

Read

```
db.collection.find(<filtre>, <projection>);
```

Équivalent SQL :

```
SELECT <champs_projetés> FROM db.collection WHERE
<conditions_du_filtre>;
```

Update

```
db.collection.update[One | Many](<filtre>, <update>,
<arguments_additionnels>);
```

Équivalent SQL :

```
UPDATE db.collection SET <champ_update_1> = <valeur_update_1>, ... WHERE
<conditions_du_filtre>; -- Les arguments additionnels servent au filtre
```

Delete

```
db.collection.delete[One | Many](<filtre>);
```

Équivalent SQL :

```
DELETE FROM db.collection WHERE <conditions_du_filtre>;
```

Ressources complémentaires

- [Introduction au JavaScript](#) - Cédric FONTAINE
- [BSON Types](#) - MongoDB Docs
- [Online Regex Tester](#) - Pour comprendre la puissance des expressions régulières