



Administration et Développement Mongo DB

Support de cours

Réf. T260-010

Réf. T260-020





Module 0

A propos de cette formation

Votre formateur

- Son nom
- Ses activités
- Ses domaines de compétence
- Ses qualifications et expériences pour ce cours

Votre formation - Présentation

- Description
 - Initiation à la manipulation de la base NO-SQL MongoDB pour insérer, modifier et extraire des données faiblement structurées.
- Profil des stagiaires
 - Architectes, concepteurs, développeurs
- Connaissances préalables
 - Connaissances générales des bases de données
- Objectifs à atteindre
 - Comprendre les principes et l'architecture d'une base MongoDB
 - Installer, configurer une base MongoDB
 - Effectuer des opérations de lecture, écriture et mise à jour de données
 - Extraire des données complexes

Votre formation - Programme

- Introduction
- Installation
- Sécurité de la base
- Console Mongo
- JavaScript Array et Object
- Requêtes simples
- Opérations d'écriture de données
- Agrégation de données
- Fonctions stockées
- Réplication de données
- Distribution de données
- Application Java
- API Morphia

Votre formation – Ressources à votre disposition

- Le présent support de cours
- Le support de référence
 - MongoDB Cookbook (Second Edition), Packt Publishing
- La documentation officielle en ligne
 - <https://docs.mongodb.com/>

Tour de table – Présentez-vous

- Votre nom
- Votre société
- Votre métier
- Vos compétences dans des domaines en rapport avec cette formation
- Les objectifs et vos attentes vis-à-vis de cette formation

Logistique

- Horaires de la formation
 - 9h00-12h30
 - 14h00-17h30
- Pauses



Module 1

Introduction



Base NoSQL de type « document »

- Un enregistrement est appelé « Document »
 - Objet indexé de type JSON

```
{
  nom : "TROADEC",
  prenom : "No1wenn",
  age: 46
}
{
  nom : "MARTIN",
  age: 32,
  tel: [ "0213456789", "0612345789" ]
}
```

Document

- Les objets sont fréquemment utilisés dans de nombreux langages de programmation
- L'imbrication des données (tableaux / objets) réduit les opérations de jointure
- Les données non structurées sont plus souples d'utilisation et plus proches des données concrètes des applications

Caractéristiques principales

- Gain en rapidité
 - Nombre d'appels à la base réduit par imbrication de données
 - Volume échangé à chaque appel plus important
 - Indexation des données
 - > Aussi possible à l'intérieurs des données imbriquées
- Support des opérations de lecture-écriture CRUD
 - Recherche par « ressemblance »
 - > Les requêtes sont des objets décrivant la structure des objets recherchés
 - Agrégation de données
 - Recherches avancées (Valeur, Contenu, Expression Régulière)

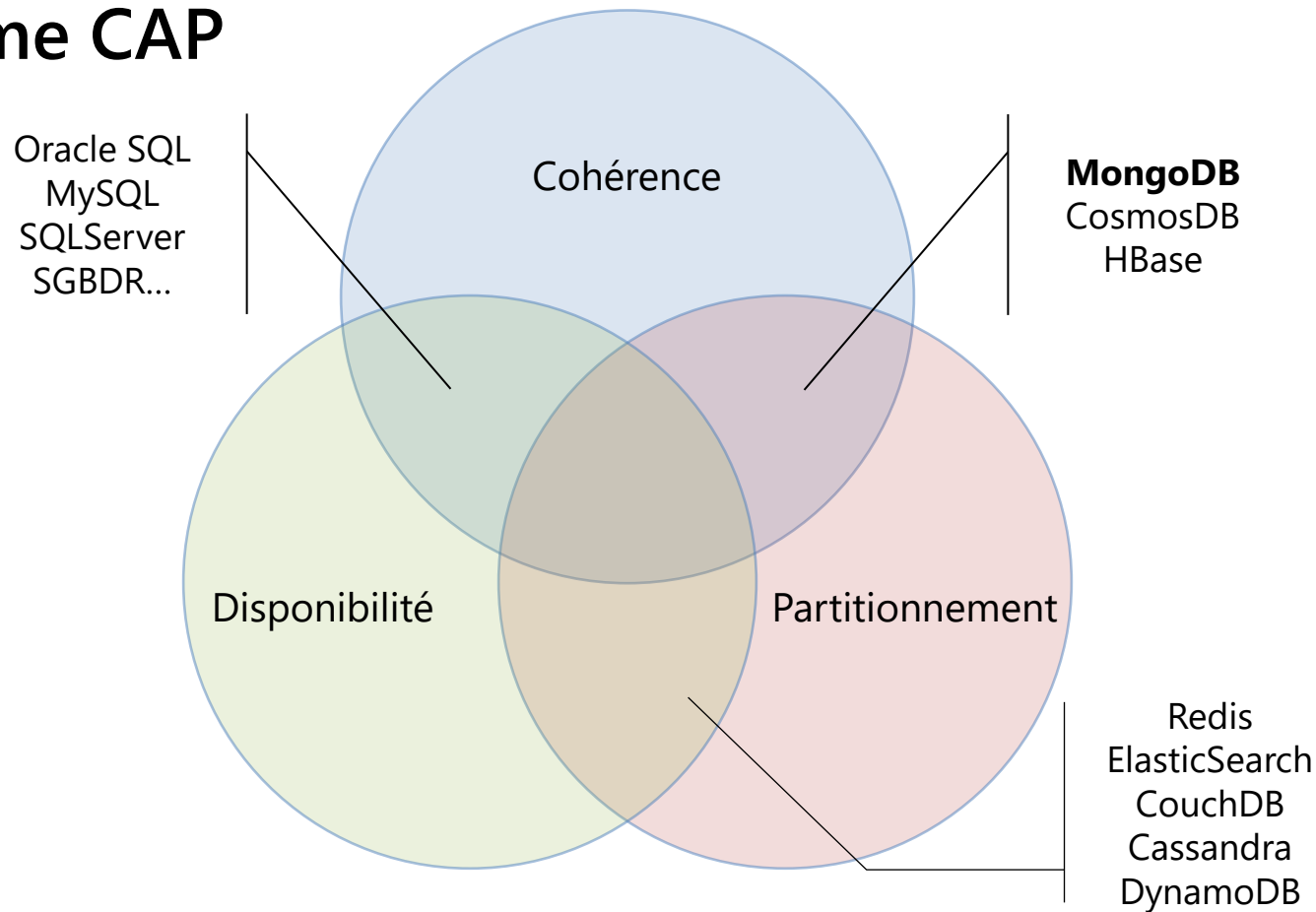
Caractéristiques principales

- Haute disponibilité (Replica Set)
 - Groupe de serveurs MongoDB
 - Redondance de données
 - Gestion automatique des erreurs
 - Meilleure disponibilité des données (lecture/écriture)
- Données distribuées (Sharding)
 - Groupe de serveurs MongoDB
 - Répartition d'un jeu de données sur plusieurs serveurs
 - Définition d'une clé de répartition
 - Amélioration des temps de réponse en lecture

Théorème CAP

- Cohérence
 - Deux demandes identiques reçoivent exactement la même réponse
- Disponibilité
 - Toutes les demandes reçoivent une réponse
- Partitionnement
 - Chaque partition de données doit répondre en autonomie à une demande
- Selon le théorème CAP (CDP), seules deux contraintes peuvent être garanties simultanément dans un système distribué en réseau.
- MongoDB garanti la Cohérence et le Partitionnement
 - SGBDR garantissent la Cohérence et la Disponibilité

Théorème CAP



Comparaison avec le modèle relationnel

- Atomicité des transactions document par document
- Pas de transactions sur plusieurs collections
- MongoDB répond à la problématique d'extensibilité des bases
- L'intégrité des transactions plus large repose sur l'application et non la base MongoDB

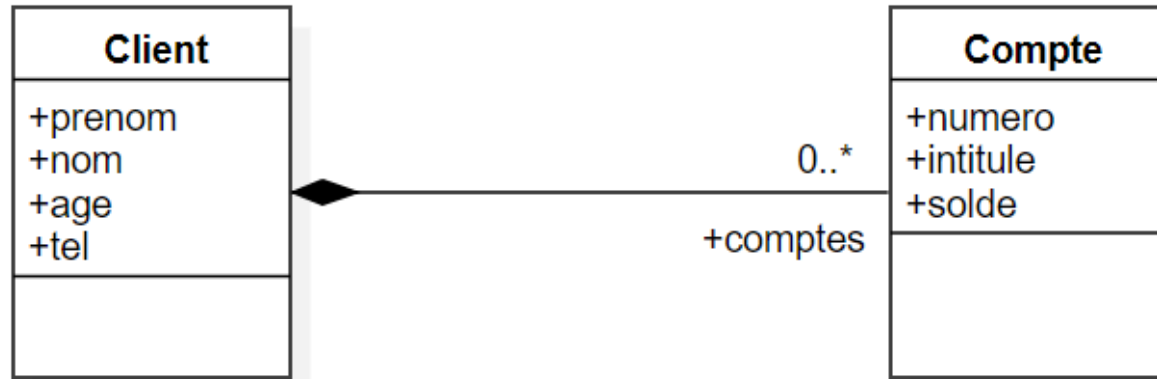
Comparaison avec le modèle relationnel

SQL	Mongo
Schema	Database
Table	Collection
Ligne	Document (Objet BSON)
Colonne	Attribut
Index	Index
Jointure	Encapsulation de documents Opérateur \$lookup
Clé primaire	Attribut _id
Fonctions de groupe	Aggrégations
Vue	View (lecture seule)

Conception du modèle

- Structure des documents
 - Choisir entre encapsulation et références
 - Choisir entre des requêtes
 - > moins nombreuses, plus complexes et plus volumineuse ; ou
 - > Plus nombreuses, plus simples et plus légères
- Encapsulation
 - Relation de composition forte
 - Petites informations qui n'évolue pas ou très rarement (tags)
- Références
 - Objets > 16 Mo
 - Liaisons multiples

Conception du modèle



Conception du modèle

Encapsulation

```
{
  nom: "Troadec",
  prenom: "Nolwenn"
  comptes: [{
    numero: "12345678",
    intitule: "Compte Courant",
    solde: 0
  }, {
    numero: "23456789",
    intitule: "Livret A",
    solde: 0
  }]
}
```

Référence

```
{
  nom: "Troadec",
  prenom: "Nolwenn"
  comptes: ["12345678", "23456789"]
}
```

```
{
  numero: "12345678",
  intitule: "Compte Courant",
  solde: 0
}
{
  numero: "23456789",
  intitule: "Livret A",
  solde: 0
}
```

Bases de données et Collections

- Une Base de données contient des Collections
- Une Collection contient des documents
- Les bases et les **collections** sont **créées implicitement** lorsqu'on y accède pour la première fois

```
mongo
> use clientsDB
> db.clients.save( { nom : "TROADEC", prenom : "Nolwenn" } )
> db.clients.save( { nom : "LENOIR", prenom : "Marc" } )
> db.clients.find()
> db.clients.find( { nom : "TROADEC" } )
```

Document

- Notation BSON

- Equivalent à la notation JSON avec plus de types de données disponibles

```
{ nom:"TROADEC", prenom:"Nolwenn",  
  adresse:{ rue:"10 quai du port", code:29000, ville:"Quimper" } }
```

```
mongo> db.clients.find( { nom : "TROADEC" } ).pretty()  
{  
  "_id" : ObjectId("4f16fd67d1e2d3237103f31e"),  
  "nom" : "TROADEC",  
  "prenom" : "Nolwenn",  
  "adresse" : {  
    "rue" : "10 quai du port",  
    "code" : 29000,  
    "ville" : "Quimper"  
  }  
}
```

Propriétés

- **_id** est réservé pour la clé primaire du document
 - Valeur unique dans la collection
 - Non modifiable
 - Type ObjectId, par convention (tout type sauf tableau)
- Le nom de la propriété ne doit pas commencer par le caractère \$
- Le nom de la propriété ne doit pas contenir de point (.)

Type BSON

Type	Alias	Console mongo
Double	"double"	123 ; 123.45
String	"string"	"texte"
Object	"object"	{ ... }
Array	"array"	[...]
Integer 32-bit	"int"	NumberInt("25")
Integer 64-bit	"long"	NumberLong("2555555000005")
ObjectId	"objectId"	ObjectId("4f16fd67d1e2d3237103f31e")
Boolean	"bool"	true ; false
Date	"date"	ISODate("2012-12-19T06:01:17.171Z")

Validation des documents

- Disponible depuis la version 3.2
- Définition d'un schéma de données lors de la création d'une collection
 - Champs obligatoires
 - Typage des champs
 - Conditions appliquées aux valeurs

Validation des documents

```
db.createCollection("clients", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: [ "nom" ],
      properties: {
        nom: {
          bsonType: "string", description: "obligatoire, doit être du texte"
        },
        prenom: {
          bsonType: "string", description: "doit être du texte"
        },
        age: {
          bsonType: "int",
          minimum: 7, maximum: 97,
          exclusiveMaximum: false,
          description: "doit être un nombre entier compris entre 7 et 97"
        }
      }
    }
  }
})...
```

Client Mongo et API

- Manipulation des données par programmation
 - Langage JavaScript dans la Console MongoDB
 - API pour les principaux langages (Java, C#, Python, etc.)
- Compass est l'application cliente fournie par Mongo pour manipuler les données d'une base
- Il existe d'autres produits développés par la communauté Mongo
 - Robot 3T (anciennement RoboMongo)
 - Studio 3T (anciennement MongoChef)



Module 2

Installation

MongoDB

- Community Server
 - Utilisation Gratuite
 - Pas de limitation de la base
 - Outils de requêtage (Compass)
- Enterprise Server
 - Cryptage des données stockées
 - Authentifications étendues
 - Outils d'administration (Ops Manager)
 - Outils de modélisation et requêtage étendus (Compass)
 - Licences commerciales

Moteurs de stockage

- WiredTiger
 - Disponible à partir de la version 3.0
 - Possibilités de crypter les données
 - Amélioration des performances
- In-Memory
 - Disponible à partir de la version 3.2
 - Réduction des opérations de lecture/écriture disques
 - Accélération des opérations CRUD
- MMAPv1
 - Moteur historique
- GridFS
 - Stockage de fichiers volumineux (Images, PDF, etc.)

Installation

- <http://www.mongodb.org>
- Télécharger et installer
- Ajouter le répertoire bin/ à la variable d'environnement PATH
- Créer un répertoire de données
 - Par défaut, C:\data\db

Démarrer le serveur

- mongod, Démarrer le serveur

```
mongod --dbpath d:\test\mongodb\data
```

- mongo, Ouvrir la console

- Commande help

```
mongo --shell
```

```
mongo --shell banque sample/mongodb-banque.js
```


Service Windows

- Créer un fichier de configuration mongod.conf
 - Format YAML
 - Pas de tabulation dans ce fichier, espace uniquement
 - Les répertoires doivent être préalablement créés (db, log, etc.)

```
systemLog:
  destination: file
  path: c:\data\log\mongod.log
  logAppend: true
storage:
  dbPath: c:\data\db
  journal:
    enabled: true
net:
  bindIp: 127.0.0.1
  port: 27017
```

Service Windows

- Créer, supprimer et réinstaller le service

```
mongod.exe --config "C:\mongodb\mongod.conf" --install
```

```
mongod.exe --remove
```

```
mongod.exe --config "C:\mongodb\mongod.conf" --reinstall
```

```
mongod.exe --config "C:\mongodb\mongod.conf" --install  
--serviceName mongo-27001 --serviceDisplayName "MongoDB 1"
```



Module 3 Sécurité

Compte Adminitrateur

- Création du compte administrateur dans la base « admin »
 - Tous les comptes utilisateurs sont définis dans la base « admin »
 - Le compte admin doit être créé avant d'activer l'authentification
- Rôles root regroupe plusieurs rôle d'administration
 - `userAdminAnyDatabase`, `dbAdminAnyDatabase`, `readWriteAnyDatabase`

Compte Adminitrateur

```
use admin
db.createUser(
  {
    user: "root",
    pwd: "motdepasse",
    roles: [{ role: "userAdminAnyDatabase", db: "admin" }]
  }
)
```

Sécurisation d'accès

- Activer l'authentification au démarrage du serveur ou dans le fichier de configuration du serveur (mongod.conf)

```
mongod --auth
```

```
security:  
  authorization: enabled
```

Connexion avec authentification

- Ouvrir la console avec le compte administrateur

```
mongo -u "root" -p "motdepasse" --authenticationDatabase "admin"
```

Authentification dans la console

- Authentification après la connexion

```
mongo
```

```
use admin
```

```
db.auth("root","motdepasse")
```


Création de comptes utilisateurs

- Ajout d'un nouvel utilisateur dans la base (admin) de gestion des comptes
 - Attribution des rôles read et readWrite

Création de comptes utilisateurs

```
use admin

db.auth("root","motdepasse")

db.createUser({
  user: "formation",
  pwd: "password",
  roles: [{
    role: "readWrite", db: "banque"
  },{
    role: "read", db: "association"
  }]
})
```

Module 4

Console Mongo



Console MongoDB

- Console d'administration et de manipulation des données
- Support du langage JavaScript
- Ouvrir la console à partir d'une fenêtre de commandes

```
mongo banqueDB
```

```
mongo --host localhost --port 27017
```

Objet db

- Sélection d'une base de donnée avec la commande use
 - La base sélectionnée est accessible via la variable implicite db

```
use clientDB  
db.getName();  
db.help();
```

Objet db

<code>db.getCollectionNames()</code>	Liste des collections
<code>db.getCollectionInfos()</code>	Liste détaillée des collections
<code>db.version()</code>	Version MongoDB
<code>db.stats()</code>	Statistique de la base (nb collections...)
<code>db.serverStatus()</code>	Etat détaillé du serveur
<code>db.getName ()</code>	Nom de la base

Objets db.collection

find()	Liste des objets d'une collection
insert()	Création d'objets
save()	Enregistrement d'objets
remove()	Suppression d'objets
count()	Nombre d'objet dans la collection

```
db.clients.find()  
db.clients.find().pretty()  
db.clients.count()
```

JavaScript

- Langage de programmation de la console Mongo
- Ecriture de scripts utilisant les éléments courants des langages de programmation
 - Variables
 - Types de données
 - Expressions
 - Instructions et blocs d'instructions
 - Boucles et conditions
 - Fonctions
 - Tableau, liste et objets

Exécution d'un script

- Les scripts doivent être écrits en JavaScript et peuvent être enregistrés dans des fichiers .js
- Exécution en passant le fichier de script en paramètre de la commande mongo.exe
- Exécution avec la fonction load() depuis le Shell

```
mongo mon-script.js  
  
> load("mon-script.js")
```

Sauvegarde et Restauration BSON

- Sauvegarde et restauration des données « brutes » BSON
 - mongodump pour sauvegarder les données
 - mongorestore pour restaurer les données
- Possibilité de sauvegarder une collection, une base ou toutes les bases
- Mécanisme d'authentification identique à la connexion au shell mongo
 - L'utilisateur doit avoir les droits readWrite et dbAdmin sur la base

Sauvegarde et Restauration BSON

```
mongodump --db association
mongodump --db association --collection ateliers
          -u root -p motdepasse --authenticationDatabase admin

mongorestore --drop dump/association/animateurs.bson
mongorestore --host 192.168.19.52 --db adherent2db dump/adherentdb
```

Sauvegarde et Restauration JSON ou CSV

- Sauvegarde et restauration des données au format JSON ou CSV
 - `mongoexport` pour sauvegarder les données
 - `mongoimport` pour restaurer les données
- Possibilité de sauvegarder une collection, une base ou toutes les bases
- Mécanisme d'authentification identique à la connexion au shell mongo
 - L'utilisateur doit avoir les droits `readWrite` sur la base

Sauvegarde et Restauration JSON ou CSV

```
mongoexport -d banquedb -c clients -o clients.json
```

```
mongoimport -d banquedb -c clients --upsert --file clients.json
```

```
mongoimport --db adherentdb --collection adherents_inscriptions --type csv  
--headerline --file adherents.csv
```



Module 5

JavaScript Array et Object

Collections

- Les collections peuvent être référencées par des variables de type tableau ou objet générique
 - Un **tableau** (Array) référence une liste de valeurs positionnées et accessibles à l'aide d'un index
 - Un **objet générique** (Object) contient une liste de valeurs référencé par un identifiant appelé « propriété »
- Les tableaux et objets génériques sont **hétérogènes** et **dynamiques**
 - Une collection peut référencer des éléments de types différents
 - La dimension d'une collection évolue en fonction des ajouts et suppressions d'éléments

Création d'un tableau

- Il existe 3 syntaxes différentes pour créer un tableau
 - Instanciation seule du type Array
 - Instanciation du type Array et initialisation du contenu
 - Utilisation de l'expression littérale [] précisant la liste des valeurs

```
var tab1 = new Array();  
var tab2 = new Array(12,"mot",true,uneVariable);  
var tab3 = [12,"mot",true,uneVariable];
```


Accès aux données d'un tableau

- L'accès aux éléments d'un tableau utilise la notation crochets []
- Le premier élément du tableau est positionné à l'index 0
- Le nombre d'éléments contenus est accessible par la propriété `length`
 - `length` désigne aussi la première position disponible

Accès aux données d'un tableau

```
var tab = [12,"mot",true,uneVariable];
tab[2] = false;
alert(tab.length);      // 4
alert(tab[0]);           // 12
alert(tab[2]);           // false
alert(tab[4]);           // undefined (hors limites du tableau)
tab[5] = "tableau dynamique";
alert(tab[5]);           // "tableau dynamique"
alert(tab[4]);           // undefined (position non initialisée)
alert(tab.length);       // 6
```

Parcourir le contenu d'un tableau

- Plusieurs structures de contrôle permettent de parcourir les éléments d'un tableau
 - La boucle for utilisée avec une variable indice parcourt chaque position du tableau, y compris les positions vides (undefined)
 - La boucle for ... in ne parcourt que les positions non vides

Parcourir le contenu d'un tableau

```
var tab = ["A","B","C"];
tab[25] = "Z";

for(var i = 0; i < tab.length; i++){
    alert(i+"-"+tab[i]); // 0-A 1-B 2-C 3-undefined 4-undefined ... 25-Z
}
for(var i in tab){
    alert(i + "-" + tab[i]); // 0-A 1-B 2-C 25-Z
}
for(let val of tab){           // ES6
    alert(val);                 // A B C Z
}
```

Fonctions d'itération

- Certaines fonctions du type Array permettent d'effectuer une même action sur chacun des éléments d'un tableau sans écrire de boucle de parcours
 - La fonction `forEach` appelle une fonction passée en paramètres pour chaque valeur du tableau
 - Les autres fonctions sont `filter()`, `every()`, `map()`, `some()`, `reduce()`, `reduceRight()`

Fonctions d'itération

```
function minuscule(valeur, position, tableau){
    if(typeof(valeur) == "string"){
        tableau[position] = valeur.toLowerCase();
    }
}

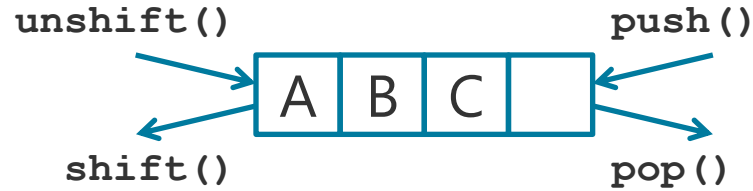
function voyelle(valeur, position, tableau){
    var txt = (typeof(valeur)=="string")?valeur.toLowerCase():"";
    if(txt=="a"||txt=="e"||txt=="i"||txt=="o"||txt=="u"||txt=="y"){
        return true;
    }
    return false
}

var tab = ["A","B","C","D","E"];
tab.forEach(minuscule); // ["a","b","c","d","e"]
var tabVoyelle = tab.filter(voyelle); // ["a","e"]
```

Gestion de files et de piles

- Les données peuvent être ajoutés et supprimés en début et en fin de tableau avec 4 fonctions
 - `push()` ajoute un élément en dernière position et retourne la nouvelle longueur du tableau
 - `pop()` retourne et supprime l'élément en dernière position dans le tableau
 - `unshift()` ajoute un élément en première position et décale les autres éléments d'une position
 - `shift()` retourne et supprime l'élément en première position
 - > Les autres éléments sont décalés d'une position pour que la position 0 correspondent au premier élément du tableau

Gestion de files et de piles



```
var tab = ["A","B","C"];  
tab.push("D");           // ["A","B","C","D"]  
var elem1 = tab.shift(); // ["B","C","D"]  
alert(elem1);           // "A"  
var elem2 = tab.pop();   // ["B","C"]  
alert(elem2);           // "D"  
tab.unshift("A");        // ["A","B","C"]
```


Sélectionner et extraire des éléments

- Certaines fonctions permettent de manipuler des sous-parties de tableau
 - slice() retourne un tableau contenant tous les éléments d'une position de début à une position de fin exclue
 - splice() supprime et retourne tous les éléments d'une position de début à une position de fin exclue

```
var tab = [8, 6, 7, 5, 3, 0, 9];  
var tabSlice = tab.slice(2,4);  
alert(tabSlice);                // 7,5  
var tabSplice = tab.splice(1,4); // tab = [8,3,0,9]  
alert(tabSplice);               // 6,7,5
```

Trier un tableau

- La fonction `sort()` trie les données d'un tableau par ordre alphabétique des valeurs converties en chaînes de caractères
 - Le tri est sensible à la casse
 - Le tri peut être configuré en ajoutant une fonction de comparaison

```
var tab = [8,16,7,25,3,10,9];

function compNombre(a,b){
    return a - b;
}

tab.sort(); // [10,16,25,3,7,8,9] tri alphabétique par défaut
tab.sort(compNombre); // [3,7,8,9,10,16,25] tri numérique
```

Objet générique

- Les objets génériques sont des collections de données dont les valeurs ne sont pas positionnées par un index mais référencées par un identifiant appelé « propriété »
- Les objets génériques sont des variable de type Object

Objet générique

- Il existe 2 syntaxes différentes pour créer un objet
 - Instanciation du type Object
 - > Le terme « instanciation » désigne l'opération de création d'un objet
 - > L'opérateur new crée un nouvel objet
 - > Les objets créés sont dynamiques et permettent l'ajout de nouvelles propriétés
 - Utilisation de l'expression littérale { } précisant la liste des propriétés sous forme de liste de paires clé-valeurs
- Les propriétés d'un objet peuvent être de types différents
 - Lorsqu'une propriété est de type fonction, on la nomme méthode
- Lorsque l'objet est créé, les propriétés et méthodes sont accessibles avec la notation pointé (.)
- Les propriétés et méthodes disponibles pour l'ensemble des types prédéfinis forment l'API (Application Programming Interface)

Objet générique

```
var pers2 = {  
  nom : "Troadec",  
  prenom : "Nolwenn",  
  dateNaiss : new Date(1960, 4, 12),  
  age : function(){  
    var _ajd = new Date().getTime();  
    return ...;  
  },  
  adresse : {  
    rue : "12 Cours des 50 Otages",  
    cp : 44000,  
    ville : "Nantes"  
  }  
};  
alert(pers2["nom"]);           // Troadec  
alert(pers2.age());           // 52  
alert(pers2.adresse.ville);   // Nantes
```

Tableau de données complexes

- Une structure de données équivalentes à une feuille de calcul ou une table de base de données peut être reconstituée en mémoire à partir d'un tableau (Array) contenant des objets génériques (Object)
 - Les objets correspondent aux lignes du tableau dont chacune des colonnes correspond à une propriété des objets génériques

	nom	prenom
[0]	Troadec	Nolwenn
[1]	Durand	Marc
[2]	Leblanc	Marie

Tableau de données complexes

```
var pers1 = {  
    nom : "Troadec",  
    prenom : "Nolwenn"  
};  
var pers2 = {  
    nom : "Durand",  
    prenom : "Marc"  
};  
var pers3 = {  
    nom : "Leblanc",  
    prenom : "Marie"  
};  
var personnes = [pers1, pers2, pers3];  
  
alert(personnes.length + " personnes"); // 3 personnes
```



Module 6

Recherches / Requêtes

Create Read Update Delete

- MongoDB : insert(), find(), update(), remove()
- SQL : Insert, Select, Update, Delete
- Rest : post, get, put, delete

Rechercher un objet

- `findOne()` retourne le premier objet d'une collection
 - Paramétrage la recherche en indiquant les valeurs recherchées
 - Choix des propriétés à retourner

```
mongo> db.clients.findOne({nom:"Troadec"})
```

```
{ "_id" : ObjectId("554a195b7c5a46a37f27ce54"), "nom" : "Troadec",  
  "prenom" : "Nolwenn", "comptes" : [ "12345678", "23456789" ] }
```

```
mongo> db.clients.findOne({nom:"Troadec"}, {nom:true, _id: false})
```

```
{"nom" : "Troadec"}
```

Rechercher des objets

- `find()` retourne tous les objets d'une collection

```
mongo> db.clients.find()
```

```
{ "_id" : ObjectId("554a195b7c5a46a37f27ce54"), "nom" : "Troadec",  
  "prenom" : "Nolwenn", "comptes" : [ "12345678", "23456789" ] }  
{ "_id" : ObjectId("554a195b7c5a46a37f27ce55"), "nom" : "Lenoir", "prenom"  
  : "Marc", "comptes" : [ "12345678" ] }
```

Opérateurs de comparaison

- Rechercher une valeur strictement supérieur avec `$gt`
 - Autres possibilités : `$gte` , `$lt` , `$lte` , `$eq` , `$ne` , `$in` , `$nin`.
- `$in` permet d'énumérer les valeur à rechercher

```
mongo> db.clients.find({nom: { $gte:"L", $lt:"M"}},{nom:true,
_id:false})
{ "nom" : "Lenoir" }
{ "nom" : "Lebreton" }
```

```
mongo> db.clients.find({nom: { $in:["Lenoir","Lebreton"] }})
{ "nom" : "Lenoir" }
{ "nom" : "Lebreton" }
```

Opérateurs \$regex et \$exists

- Rechercher les objets qui possèdent une propriété avec \$exists
- Rechercher des valeurs textes complexes avec \$regex

```
mongo> db.clients.find({comptes:{ $exists: false}})
{ "nom" : "Troadec", "prenom" : "Nolwenn" }
{ "nom" : "Lenoir", "prenom" : "Marc" }
```

```
mongo> db.clients.find({nom:{ $regex: /t/i}})
{ "nom" : "Troadec", "prenom" : "Nolwenn" }
{ "nom" : "Lebreton", "prenom" : "Erwan" }
```

Opérateurs logiques \$or et \$and

- Combinaison de plusieurs critères de recherche

```
mongo> db.clients.find({$or:[ {nom:{$regex:"a"}} , {prenom:{$regex:"a"}} ]})
{ "nom" : "Troadec", "prenom" : "Nolwenn" }
{ "nom" : "Lenoir", "prenom" : "Marc" }
{ "nom" : "Durand", "prenom" : "Marie" }
{ "nom" : "Lebreton", "prenom" : "Erwan" }
```

```
mongo> db.clients.find({$and:[ {nom:{$regex:"a"}} , {prenom:{$regex:"a"}} ]})
{ "nom" : "Durand", "prenom" : "Marie" }
```

Rechercher une valeur dans un tableau

- Certaines propriétés contiennent des listes de valeurs
 - MongoDB recherche indifféremment une valeur ou la présence d'une valeur dans un tableau
- `$all` permet de rechercher la présence de plusieurs valeurs dans un tableau

```
mongo> db.clients.find({comptes:"12345678"}, {_id: false})
{ "nom":"Troadec", "prenom":"Nolwenn", "comptes": [ "12345678", "23456789" ] }
{ "nom":"Lenoir", "prenom":"Marc", "comptes": [ "12345678" ] }
```

```
mongo> db.clients.find({comptes: {$all:["12345678","23456789"]}, {_id: false})
{ "nom":"Troadec", "prenom":"Nolwenn", "comptes": [ "12345678", "23456789" ] }
```

Rechercher une valeur dans un objet imbriqué

- Utilisation de la notation pointée dans le critère de recherche

```
mongo> db.clients.find({adresse:{ville: "Nantes"}})

mongo> db.clients.find({"adresse.ville":"Nantes"})
{
  "nom": "Troadec",
  "prenom": "Nolwenn",
  "adresse": {
    "rue": "10 cours des 50 Otages",
    "code": 44000,
    "ville": "Nantes"
  }
}
```


Curseur

- La fonction `find()` retourne un objet curseur permettant de parcourir un à un les objets du résultat de la requête
- `hasNext()` retourne un booléen, vrai s'il reste des objets à parcourir
- `next()` retourne l'objet suivant

```
mongo> var curseur = db.clients.find();  
mongo> while(curseur.hasNext()) printjson(curseur.next().nom);  
"Troadek"  
"Lenoir"  
"Durand"  
"Meyer"  
"Lebreton"
```

Curseur : Tri et Limite

- `sort()` permet de trier le curseur en indiquant la ou les propriétés à ordonner
 - -1 pour un tri décroissant
- `limit()` garantit un nombre maximum d'objets retournés
- `skip()` retourne les objets à partir de la position indiquée
- L'ordre d'appel doit être respecté (`sort`, `limit`, `skip`)

Curseur : Tri et Limite

```
mongo> var curseur = db.clients.find();
```

```
mongo> curseur.sort({nom:-1}).limit(3).skip(2);
```

```
mongo> while(curseur.hasNext()) printjson(curseur.next().nom);
```

```
"Lenoir"
```

```
"Lebreton"
```

```
"Durand"
```

Compter les objets

- La fonction `count()` peut être utilisée à la place de la fonction `find()` pour obtenir le nombre d'objets retournés

```
mongo> db.clients.count({nom: { $gte:"L", $lt:"M"}})
```

```
2
```



Module 7

Opérations d'écriture des données

Insertion d'objet

- `insert()` enregistre l'objet dans une collection
 - Possibilité de passer un tableau d'objets
- Génération d'un identifiant unique dans la propriété `_id`

Insertion d'objet

```
mongo> db.clients.insert([{  
  nom: "Troadec",  
  prenom: "Nolwenn",  
  comptes: ["12345678", "23456789"]  
}, {  
  nom: "Lenoir",  
  prenom: "Marc",  
  comptes: ["12345678"]  
}])
```

```
mongo> db.clients.find()
```

```
{ "_id" : ObjectId("554a195b7c5a46a37f27ce54"), "nom" : "Troadec",  
  "prenom" : "Nolwenn", "comptes" : [ "12345678", "23456789" ] }  
{ "_id" : ObjectId("554a195b7c5a46a37f27ce55"), "nom" : "Lenoir",  
  "prenom" : "Marc", "comptes" : [ "12345678" ] }
```

Modifier l'intégralité d'un objet

- La fonction `update()` permet de modifier la totalité du contenu d'un objet à l'exception de son identifiant (`_id`)
 - Par défaut, seul le premier objet vérifiant le critère est modifié

Modifier l'intégralité d'un objet

```
{
  "_id":ObjectId("554b7ad9d71425942ca8b082"),
  "nom":"Lenoir",
  "prenom":"Marc",
  "comptes":[ "12345678" ]
}
> db.clients.update({nom:"Lenoir"},{nom:"Martin",email:"marc.martin@mail.eu"})

> db.clients.find({nom:"Martin"})
{
  "_id":ObjectId("554b7ad9d71425942ca8b082"),
  "nom":"Martin",
  "email":"marc.martin@mail.eu"
}
```

Modifier les propriétés d'un objet

- `$set` permet de modifier uniquement les propriétés précisées
 - `$unset` pour supprimer un attribut
 - `$rename` pour renommer un attribut

Modifier les propriétés d'un objet

```
{
  "_id":ObjectId("554b7ad9d71425942ca8b082"),
  "nom":"Lenoir",
  "prenom":"Marc",
  "comptes":[ "12345678" ]
}
> db.clients.update({nom:"Lenoir"},{$set:{email:"mmartin@mail.eu"}})
> db.clients.find({nom:"Lenoir"})
{
  "_id":ObjectId("554b7ad9d71425942ca8b082"),
  "nom":"Lenoir",
  "prenom":"Marc",
  "email":"marc.martin@mail.eu",
  "comptes":[ "12345678" ]
}
```

Modifier une propriété tableau

- \$push permet d'ajouter une valeur à une liste de valeurs
 - \$pop , \$pull , \$pushAll , \$pullAll , \$addToSet ,

\$pop	Retire la dernière(1) ou première(-1) valeur du tableau
\$pull	Retire la valeur correspondante du tableau
\$pushAll	Ajoute une liste de valeurs
\$pullAll	Retire une liste de valeurs
\$addToSet	Ajoute une valeur si elle n'est pas déjà présente

Modifier une propriété tableau

```
{
  "_id":ObjectId("554b7ad9d71425942ca8b082"),
  ...
  "comptes":[ "12345678" ]
}
> db.clients.update({nom:"Lenoir"},{$set:{"comptes.0":"1234"}})
> db.clients.update({nom:"Lenoir"},{$push:{comptes:"2345"}})
>
db.clients.update({nom:"Lenoir"},{$pushAll:{comptes:["3456","4567"]}})
> db.clients.update({nom:"Lenoir"},{$pop:{comptes:1} })
> db.clients.update({nom:"Lenoir"},{$pull:{comptes:"2345"}})
> db.clients.find({nom:"Lenoir"})
{
  "_id":ObjectId("554b7ad9d71425942ca8b082"),
  ...
  "comptes":[ "1234","3456" ]
}
```

Modifier une propriété ou créer l'objet

- `upsert` est une option de la fonction `update` permettant de créer un nouvel objet si l'objet à modifier n'existe pas

```
> db.clients.update(  
...   {nom:"Leblanc"},  
...   {$set:{email:"mleblanc@mail.bzh"}},  
...   {upsert: true})  
  
> db.clients.find({nom:"Leblanc"})  
{  
  "_id":ObjectId("554b7ad9d71425942ca8b088"),  
  "nom":"Leblanc",  
  "email":"mleblanc@mail.bzh"  
}
```

Modifier plusieurs objets

- `multi` est une option de la fonction `update` permettant de modifier plusieurs objets vérifiant le critère en une seule instruction
 - Par défaut `{multi: false}`, modifie uniquement le premier objet vérifiant le critère
- MongoDB assure l'intégrité des modifications objet par objet
 - Si l'instruction impacte une grande quantité d'objets, d'autres instructions de lecture et d'écriture pourront s'exécuter en cours de mise à jour (avec éventuellement des risques de conflits et/ou d'incohérence des mises à jour)

Modifier plusieurs objets

```
> db.clients.update(  
...   {},  
...   {$set:{password:"123456"}},  
...   {multi: true})
```

Ajout du même mot de passe (123456) à tous les clients

Supprimer des objets

- `remove` supprime un ou plusieurs objets vérifiant le critère de sélection
- `drop` supprime la collection dans son ensemble

```
> db.clients.remove({nom:"Lenoir"})
```

```
> db.clients.remove({})
```

```
> db.clients.drop()
```



Module 8

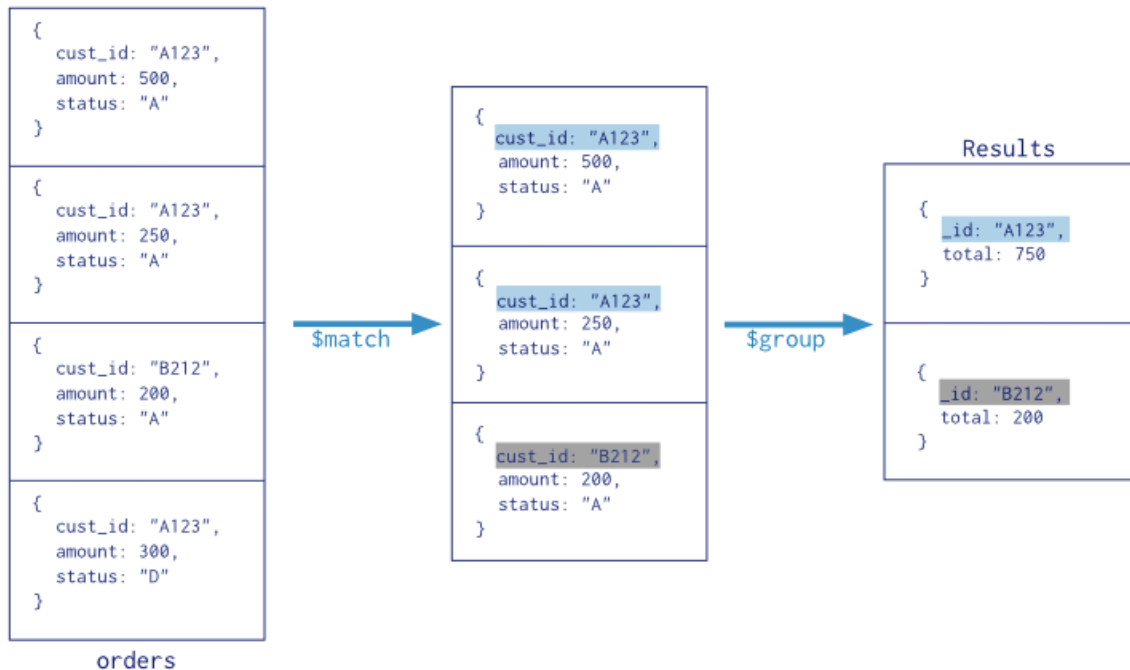
Agrégation de données

Processus d'agrégation

- Suite d'opérations sur une collection retournant un résultat
 - Filtre
 - Regroupement de données
 - Fonction de groupes (sommés, moyennes)
- La fonction aggregate() attend en paramètre un tableau d'opérations à appliquer à la collection
 - Chaque étape retourne la collection transformée

Processus d'agrégation

Collection
↓
`db.orders.aggregate([`
 `$match stage → { $match: { status: "A" } },`
 `$group stage → { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }`
 `])`



Opération d'agrégation

\$project	Sélectionne les attributs à conserver dans les objets
\$unwind	Aplatit une arborescence d'objets en une collection
\$group	Regroupe des objets en appliquant une fonction de groupe (\$sum,\$min,\$avg)
\$lookup	Effectue une jointure externe avec une autre collection de la même base
\$match	Applique un filtre sur la collection
\$sort	Trie la collection
\$skip	Ignore une partie des objets de la collection
\$limit	Limite le nombre d'objets conservés dans la collection
\$out	Ecrit le résultat dans une collection temporaire dont le nom doit être préfixé par « out_ »

Fonction de groupe

- `_id` sert ici de critère de regroupement (par numéro)
- `$first` et `$sum` sont les fonctions de groupe

```
db.clients.aggregate([
  {$unwind: "$comptes"},
  {$project: {
    numero:"$comptes.numero",
    solde:{$sum:"$comptes.operations.montant"}
  }},
  {$group: {
    _id:"$numero",
    numero:{ $first:"$numero" },
    solde:{ $sum:"$solde" }
  }},
  {$project: {
    numero:1,
    solde:1
  }},
  {$out: "out_comptes_solde"}
]);
```

Jointure

- `$lookup` effectue une jointure externe entre deux collections
 - `as` définit l'attribut de type tableau contenant les documents de la collections jointe
 - S'il n'y a pas de document joints, la valeur de l'attributs est un tableau vide

```
db.adherents.aggregate([{\n  $lookup:{\n    from: "inscriptions",\n    as: "inscriptions",\n    localField: "_id",\n    foreignField: "adherent_id"\n  }\n},{\n  $project:{\n    "inscriptions._id":0,\n    "inscriptions.adherent_id":0\n  }\n},{\n  $out: "out_adherents_inscriptions"\n}]);
```

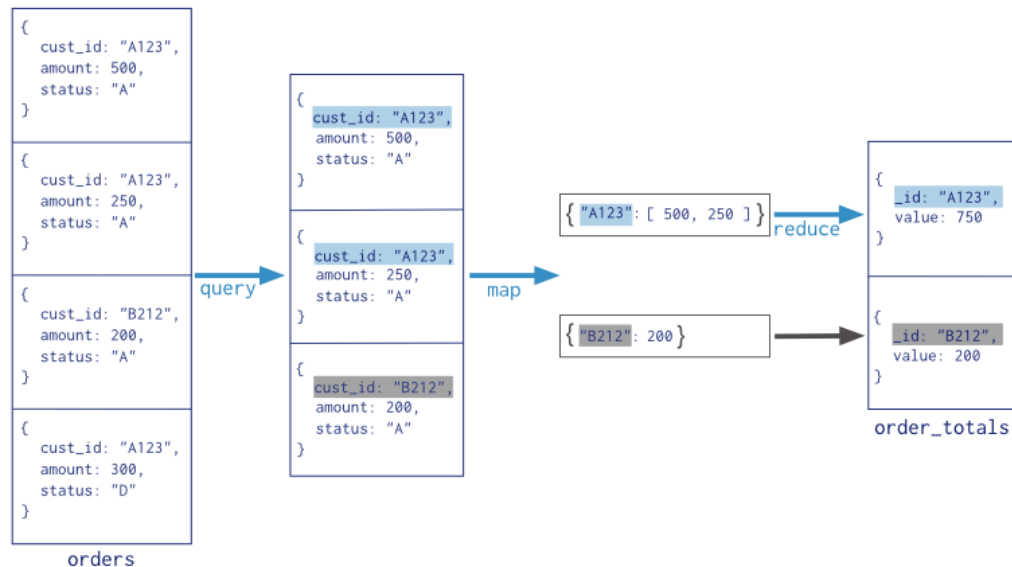
Map-Reduce

- Combinaison de fonctions appliquée à une collection retournant un résultat
 - Selection de document
 - Fusion de données

Map-Reduce

Collection
↓
db.orders.mapReduce(
 map → function() { emit(this.cust_id, this.amount); },
 reduce → function(key, values) { return Array.sum(values) },

 query → { query: { status: "A" },
 output → out: "order_totals"
})



The background features a dark grey field filled with a repeating pattern of binary code (0s and 1s). Overlaid on this is a faint, light blue flowchart. The flowchart consists of numerous rectangular boxes of varying sizes, some of which are interconnected by arrows. Some boxes contain binary strings, while others are empty. The overall aesthetic is technical and digital.

Module 9

Fonctions et procédures stockées

Fonctions stockées

- Il est possible d'enregistrer dans une collection spécifique des fonctions JavaScript pour les réutiliser dans les scripts ou les opérations de la console Mongo
- La collection doit être nommée « system.js »
- Les fonctions sont propres à chaque database

```
db.system.js.save({  
  _id : "majuscule" ,  
  value : function(texte){  
    return texte.toUpperCase();  
  }  
});
```

Fonctions stockées

- Avant de pouvoir utiliser les fonctions, elles doivent être chargées par le client Mongo
- Les fonctions peuvent être utilisées
 - Dans les scripts JavaScript comme des fonctions natives
 - Dans les opérations, requêtes ou étapes d'agrégation

```
db.loadServerScripts();

print(majuscule("texte à afficher"));

db.ateliers.insert({
  intitule: majuscule("tennis")
});
```



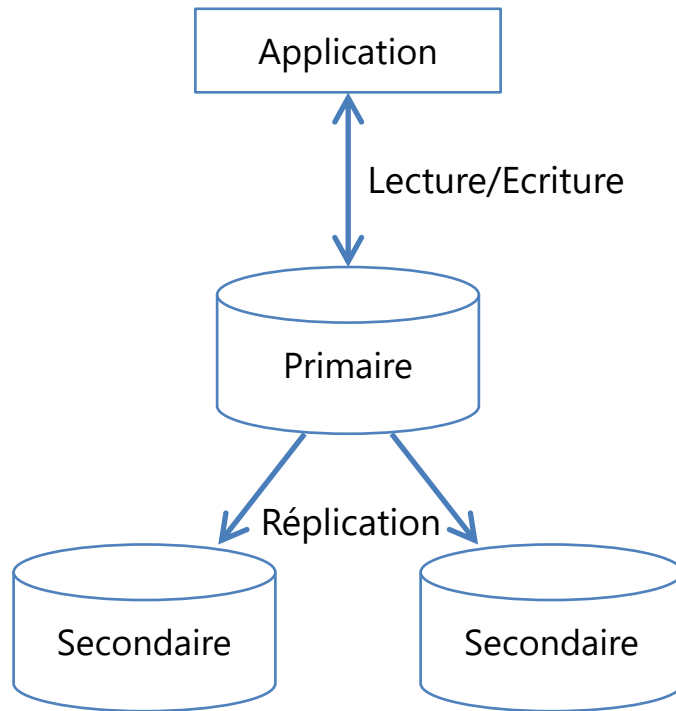

Module 10

Réplication de données

Replica Set

- Groupe de processus mongod
- Réplication du même jeu de données
- Amélioration de la disponibilité des données
- Amélioration de la tolérance aux pannes
- Une instance principale (primary)
- Plusieurs instances de sauvegarde (secondary)
- Les applications doivent se connecter à l'instance principale

Replica Set



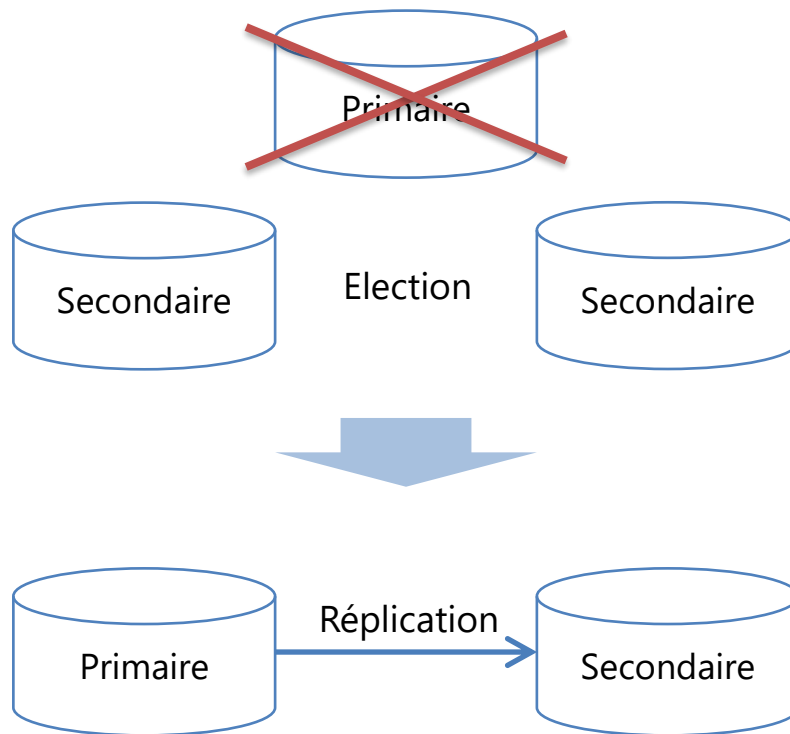
Oplog

- Chaque opération d'écriture sur l'instance principale d'un Replica Set est enregistrée dans une collection particulière appelée « oplog »
- Le Replica Set repose sur un mécanisme de réplication d'instructions
 - Les instructions du « oplog » de l'instance principale sont exécutées sur les instances secondaires
 - Il n'y a pas de réplication intégrale des données

Gestion des pannes

- L'instance principale est élue par l'ensemble des instances
 - Il est conseillé de configurer les Replica Set avec un nombre impair d'instances
- Les instances se « surveillent » mutuellement lorsque l'instance principale ne répond plus, l'une des autres instances devient instance principale après une nouvelle élection
- Lorsqu'une instance redémarre, elle reprend son mécanisme de réplication du oplog principal en repartant de sa dernière opération répliquée.

Gestion des pannes



Optimisations

- Il est possible de configurer une instance en tant qu'arbitre pour maintenir un nombre suffisant d'instances pour garantir le bon fonctionnement du processus d'élection
 - Les instances arbitres ne contiennent et ne répliquent aucune donnée
- A partir de la version 3.6, les drivers détectent les pertes de l'instance principale et relancent les opérations d'écriture après élection d'une nouvelle instance primaire
- Il est possible de configurer les applications pour qu'elles exécutent leurs opérations de lecture sur une instance secondaire.
 - Préserver l'instance primaire pour les opérations d'écriture

Configuration d'une instance

- Chaque instance doit avoir ses propres port réseau et répertoire db/
- L'attribut `repSetName` définit le groupe d'instances

```
systemLog:
  destination: file
  path: "C:/_applications/mongodb/replicaset/log/replica1.log"
  logAppend: true
storage:
  dbPath: "C:/_applications/mongodb/replicaset/replica1/db"
  journal:
    enabled: true
net:
  bindIp: 0.0.0.0
  port: 27001
replication:
  oplogSizeMB: 128
  repSetName: monReplica
```

Démarrer un replicaset

- Lancer les processus mongod
- Initialiser le replicaset avec la commande `rs.initiate()`

```
mongod --config c:\formation\rs\repl1.conf  
mongod --config c:\formation\rs\repl2.conf  
mongod --config c:\formation\rs\repl3.conf
```

```
rs.initiate(  
  { _id:'z',  
    members:[  
      { _id:1, host:'localhost:27001', priority:2 },  
      { _id:2, host:'localhost:27002', arbiterOnly:true },  
      { _id:3, host:'localhost:27003' }  
    ]  
  }  
);
```

Mode lecture seule

- La commande `setSecondaryOk()` permet d'activer l'accès en lecture des instances secondaires
 - Ajouter une priorité à 0 pour s'assurer que l'instance ne deviendra pas l'instance principale

```
rs.initiate(  
  { _id:'z',  
    members:[  
      { _id:1, host:'localhost:27001' },  
      { _id:2, host:'localhost:27002' },  
      { _id:3, host:'localhost:27003', priority:0 }  
    ]  
  }  
);  
  
db.getMongo().setSecondaryOk();
```

The background is a dark blue-grey gradient. It features a faint, repeating pattern of binary code (0s and 1s) in a lighter blue-grey color. Overlaid on this is a network diagram consisting of several rectangular nodes connected by white lines. Some nodes are simple rectangles, while others are diamond shapes. The connections form a complex web, with some nodes having multiple incoming and outgoing links. The overall aesthetic is technical and digital.

Module 10

Distribution de données

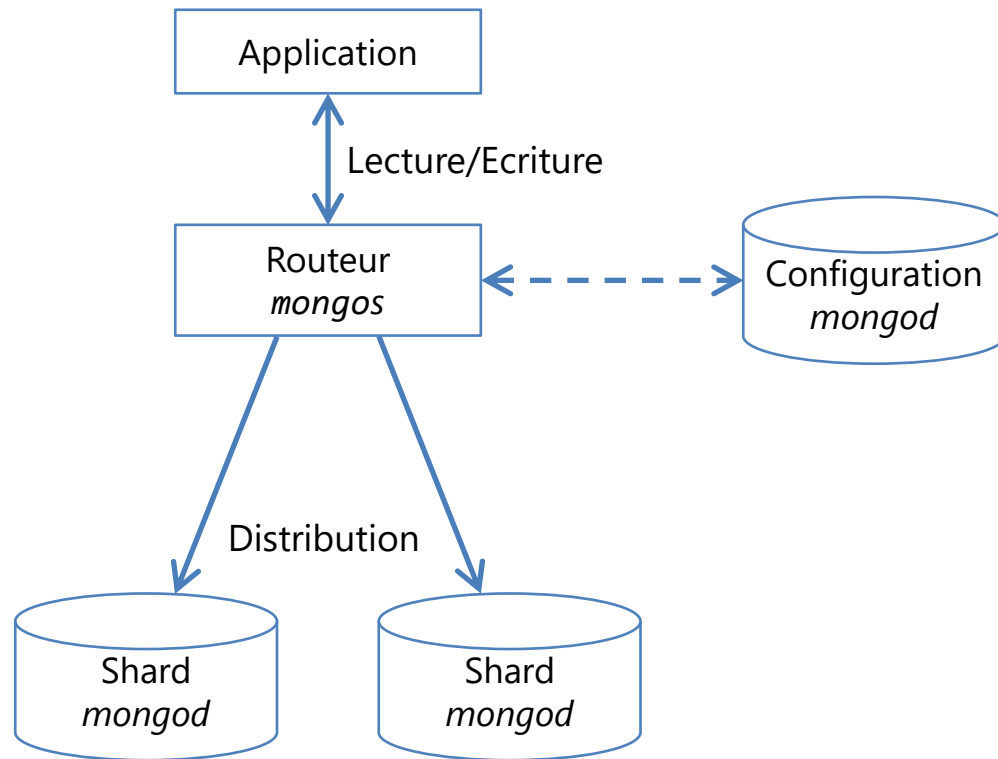
Sharding

- Le « Sharding » est un mécanisme de distribution d'un jeu de données à travers plusieurs instances et/ou machines physiques
 - Gestion d'un volume de donnée très important
 - Pas de duplication de données
- Amélioration des temps de réponse en augmentant le nombre de processeurs et les capacités mémoires sollicités pour une requête
- Distribution des données selon une clé ou la valeur d'un attribut des documents

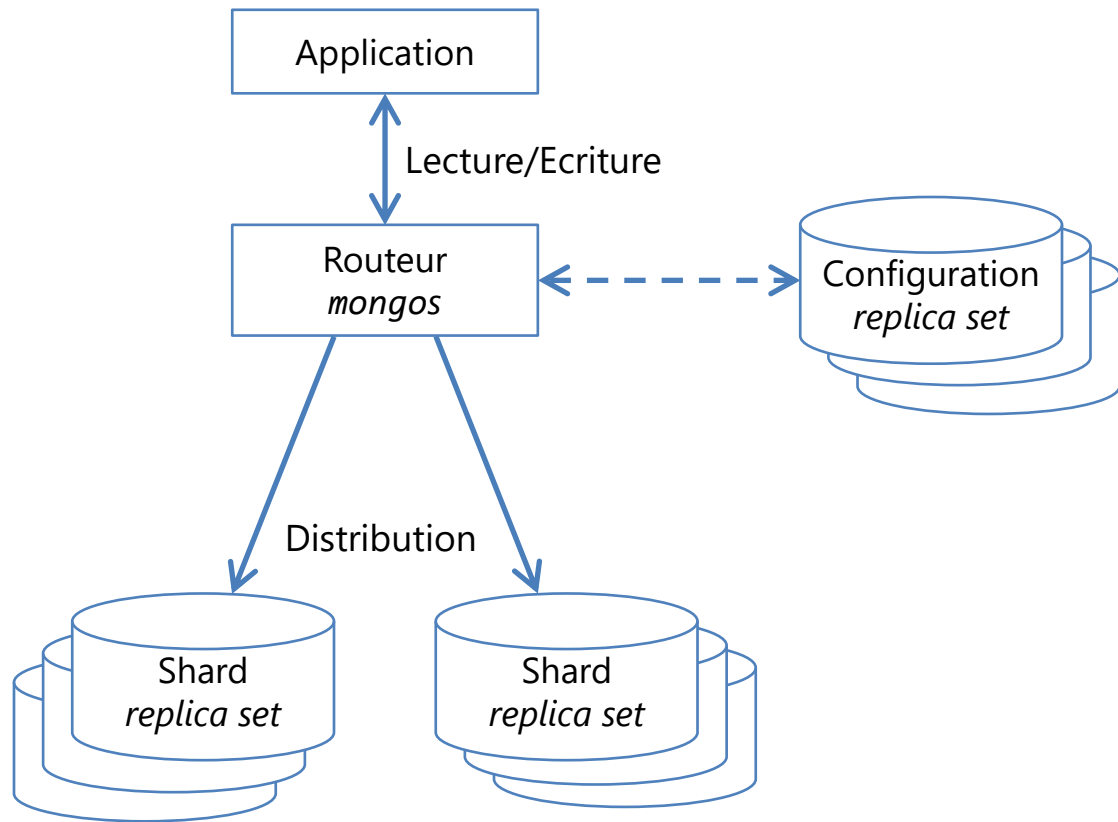
Cluster

- Un cluster MongoDB est composé de
 - Un routeur (mongos)
 - Un serveur de configuration, sans données, (mongod)
 - Plusieurs instances de distribution des données (mongod)
- Chaque instance mongod peut-être un Replica Set
- A partir de la version 3.4, le serveur de configuration doit être un Replica Set

Architecture du cluster

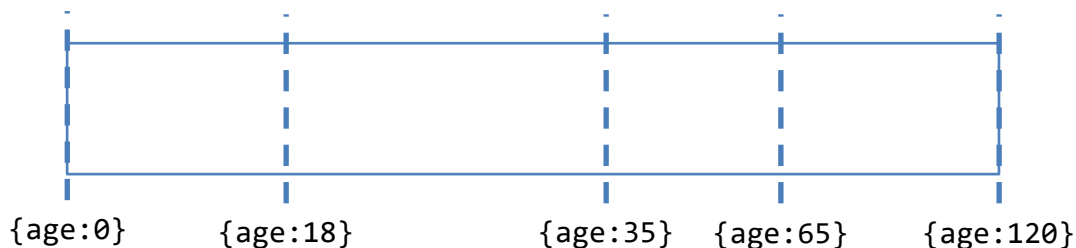


Architecture du cluster avec Replica Set



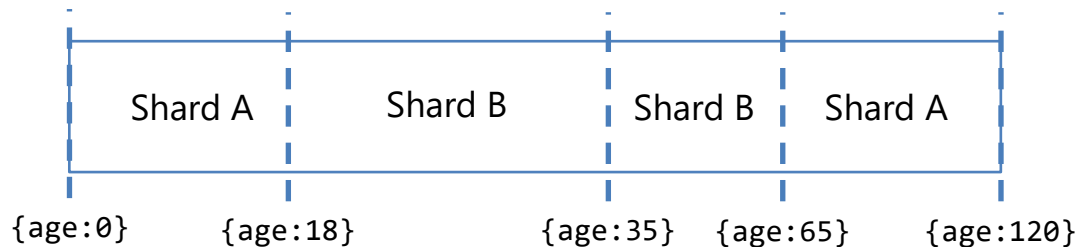
Clé de répartition

- La distribution des documents entre les instances utilise une clé de répartition
 - Attribut non modifiable et obligatoire dans tous les documents
- Le choix de la clé de répartition est définitif
 - Création d'un index pour l'attribut clé
 - Impacte les performances, la maintenabilité et l'évolutivité de la distribution des données



Bloc de données

- MongoDB divise les Shards en blocs de données (chunks)
 - 64 Mo maximum et un nombre de documents limité
 - Division automatique des blocs lorsqu'une limite est atteinte
- Chaque bloc a des limites inférieure et supérieure basées sur la clé de répartition
- Pour rééquilibrer les Shards, les blocs de données peuvent être déplacés entre les instances



Configuration du routeur

- Le routeur est une instance du service mongos
 - Doit connaître l'adresse réseau du serveur de configuration

```
systemLog:
  destination: file
  path: "D:/mongodb/log/sh/mongos.log"
  logAppend: true
net:
  bindIp: localhost
  port: 26100
sharding:
  configDB: localhost:25100
```

Serveur de configuration

- Le serveur de configuration est une instance du service mongod avec le rôle configsvr
 - La configuration est enregistrée dans un répertoire db/

```
systemLog:
  destination: file
  path: "D:/mongodb/log/sh/config.log"
  logAppend: true
storage:
  dbPath: "D:/mongodb/data/sh/config"
  journal:
    enabled: true
net:
  bindIp: localhost
  port: 25100
sharding:
  clusterRole: configsvr
```

Configuration d'un Shard

- Un Shard est une instance du service mongod avec le rôle shardsvr

```
systemLog:
  destination: file
  path: "D:/mongodb/log/sh/shard1.log"
  logAppend: true
storage:
  dbPath: "D:/mongodb/data/sh/shard1"
  journal:
    enabled: true
net:
  bindIp: localhost
  port: 27100
sharding:
  clusterRole: shardsvr
```


Démarrer un cluster

- Lancer le serveur de configuration, les shards et le routeur
- Après connexion au routeur, ajouter les shards

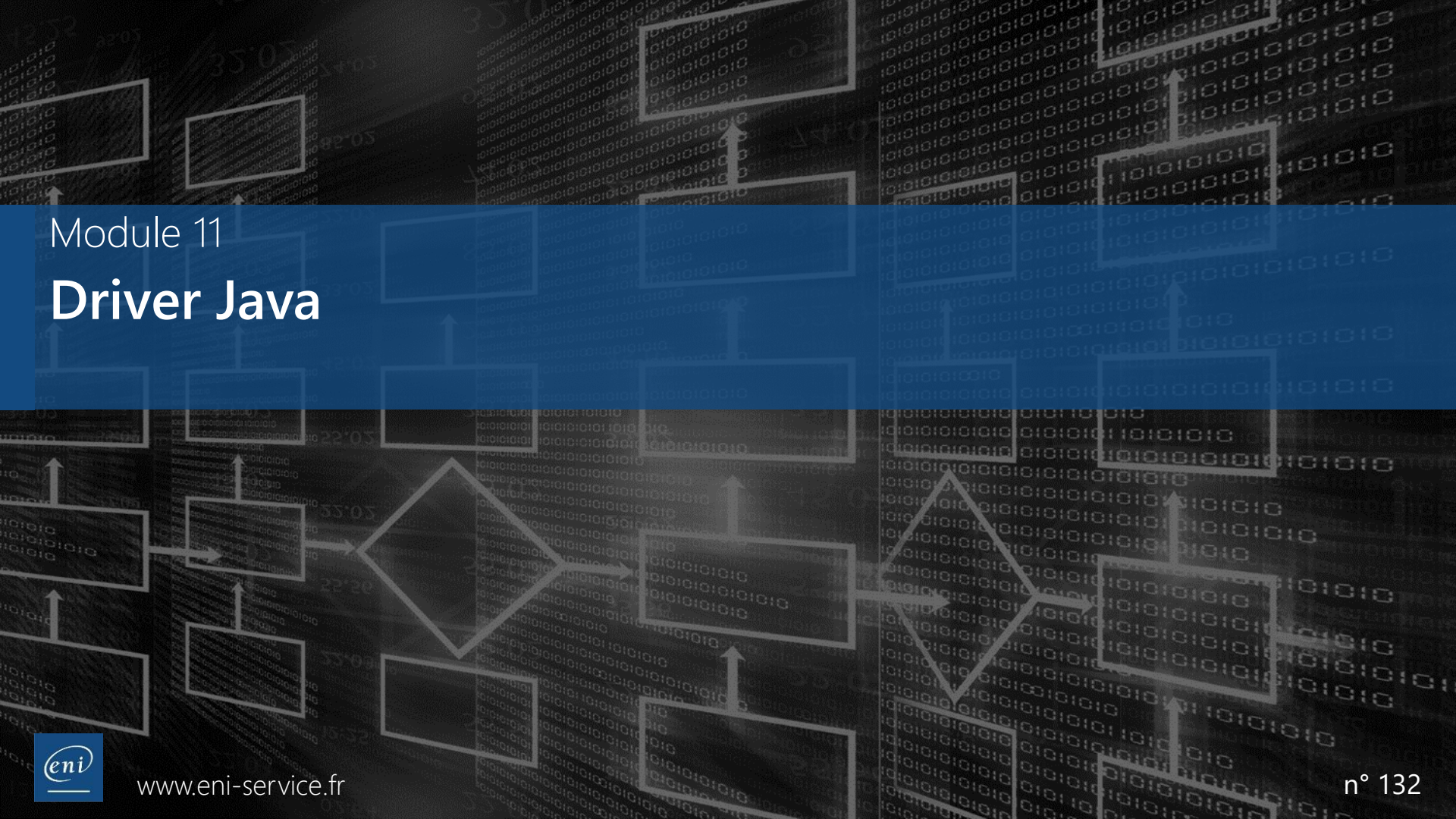
```
mongod --config c:\formation\sh\shard1.conf  
mongod --config c:\formation\sh\shard2.conf  
mongod --config c:\formation\sh\config.conf  
mongos --config c:\formation\sh\mongos.conf
```

```
> mongo --port 26100  
  
sh.addShard("localhost:27100")  
sh.addShard("localhost:27101")
```

Supervision

- Les critères de distribution des données dans les shards sont définis dans la base config

```
use config  
db.chunks.find()
```

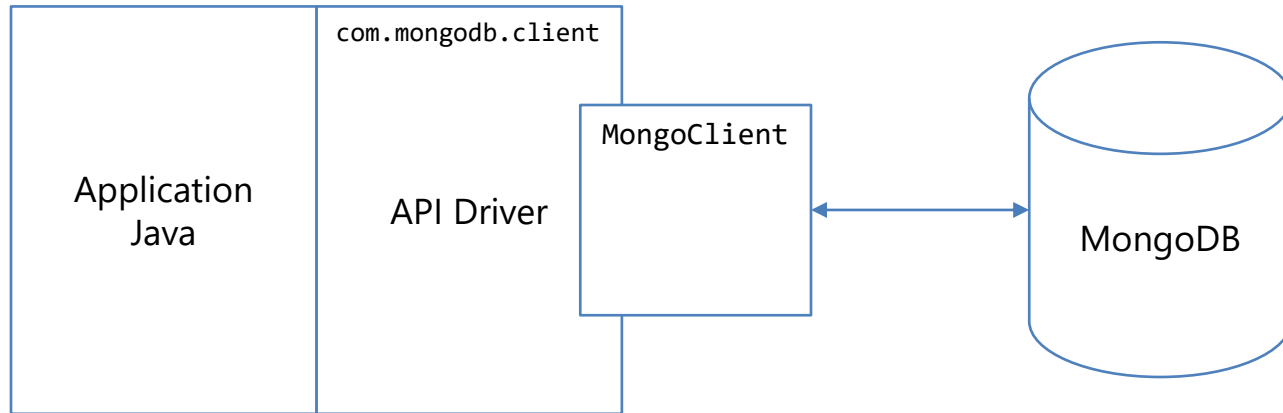


Module 11

Driver Java

Architecture

- MongoDB propose un driver Java permettant d'établir la connexion entre une application Java et une base MongoDB



Classes principales

MongoClient	Connexion au serveur MongoDB
MongoClientURI	URL de connexion au serveur MongoDB
MongoDatabase	Sélection de la base courante
MongoCollection	Manipulation d'une collection d'objet
Document	Objet dans une collection
MongoCursor	Itérateur sur le résultat d'une requête

Installation

- Création d'un projet Java avec Maven et Eclipse

New Maven Project

Select project name and location

☒ Create a simple project (skip archetype selection)

☒ Use default Workspace location

Location: C:_git\workspace-2018\mongo-java\pom.xml

☐ Add project(s) to working set

Working set:

► Advanced

New Maven Project

Configure project

Artifact

Group Id: fr.eni.formation

Artifact Id: mongodb-java

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name:

Description:

Parent Project

Group Id:

Artifact Id:

Version:

► Advanced

Installation

- Ajout de la dépendance mongodb-driver

```
<dependencies>
  <dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver</artifactId>
    <version>3.6.3</version>
  </dependency>
</dependencies>
```

Connexion à une base de donnée

- Par défaut, connexion sur localhost:27017

```
MongoClient client = new MongoClient();  
MongoClient client = new MongoClient("localhost", 27017);
```

- Connexion par URI

```
MongoClientURI uri = new  
MongoClientURI("mongodb://localhost:27017");  
MongoClient client = new MongoClient(uri);
```


Connexion à une base de donnée sécurisée

- Connexion à l'aide d'une configuration `MongoCredential`

```
ServerAddress address = new ServerAddress("localhost", 27017);  
MongoCredential credential =  
    MongoCredential.createCredential("formation", "admin", "password".toCharArray());  
MongoClientOptions options = MongoClientOptions.builder().build();  
  
MongoClient client = new MongoClient(address, credential, options);
```

- Connexion par URI

```
MongoClientURI uri = new MongoClientURI(  
    "mongodb://formation:password@localhost:27017/?AuthSource=admin");  
  
MongoClient client = new MongoClient(uri);
```

Database

- La classe `MongoDatabase` donne accès à une base MongoDB
 - La création est implicite lorsqu'on accède à une base qui n'existe pas
- La suppression d'une base n'est possible que si elle a été créée par l'application

```
MongoDatabase db = client.getDatabase("banque-java");  
  
db.createCollection("clients");  
MongoCollection clients = db.getCollection("clients");  
  
db.drop();
```

Database

- L'objet Database permet de créer, parcourir ou accéder à des collections
 - Mongolterable est le résultat des recherches et requêtes
 - MongoClient sert à parcourir les objets « Iterable »

```
MongoDatabase db = client.getDatabase("banque-java");

MongoIterable<String> collections = db.listCollectionNames();
MongoCursor<String> cursor = collections.iterator();
while (cursor.hasNext()) {
    System.out.println(cursor.next());
}
```

Collection

- La méthode `getCollection()` retourne une collection à partir de son nom
 - Si la collection n'existe pas, elle sera créée lors de l'insertion d'un premier objet

```
MongoDatabase db = client.getDatabase("banque-java");  
  
MongoCollection<Document> clients = db.getCollection("clients");
```

Document

- Les objets des collections sont représentés par le type Document
 - Classe `org.bson.Document`
 - Pas de structure imposée contrairement aux classes
 - Possibilité d'utiliser des classes en associant des Codec aux collections
- Les documents sont des collections clé-valeur

Document

```
Document client1 = new Document()
    .append("nom", "Leblanc")
    .append("prenom", "Marc")
    .append("tel", Arrays.asList("0612345789", "0213456789"))
    .append("adresse", new Document()
        .append("rue", "12 impasse des Lilas")
        .append("cp", "44000")
        .append("ville", "Nantes")
    )
    .append("naissance", Date.from( LocalDate.of(1960,1,22)
        .atStartOfDay().atZone(ZoneId.systemDefault()).toInstant()
    ))
;
```

Insertion d'un objet

- Insertion d'un objet ou d'une liste d'objets avec `insertOne()` et `insertMany()`

```
Document client1 = new Document()...

clients.insertOne(client1);

List<Document> nouveauxClients = new ArrayList<>();
nouveauxClients.add(client2);
nouveauxClients.add(client3);
nouveauxClients.add(client4);

clients.insertMany(nouveauxClients);
```

Requête sur une collection

- La méthode `find()` permet de lister tous les objets d'une collection
- La classe utilitaire `Filters` contient des méthodes statiques pour filtrer la recherche
 - `eq()`, `all()`, `and()`, `exists()`, `gt()`, `lte()`, `in()`, `not()`, `regex()`, etc.
- Lecture du premier objet avec `first()` ou parcours de tous les objets avec `iterator()`

Requête sur une collection

```
Document cliA = clients.find(Filters.eq("nom", "Leblanc")).first();  
System.out.println(cliA.toJson());
```

```
MongoCursor<Document> cliL = clients.find(  
    Filters.regex("nom", "^L", "i")  
).iterator();
```

```
while(cliL.hasNext()) {  
    System.out.println(cliL.next().toJson());  
}
```

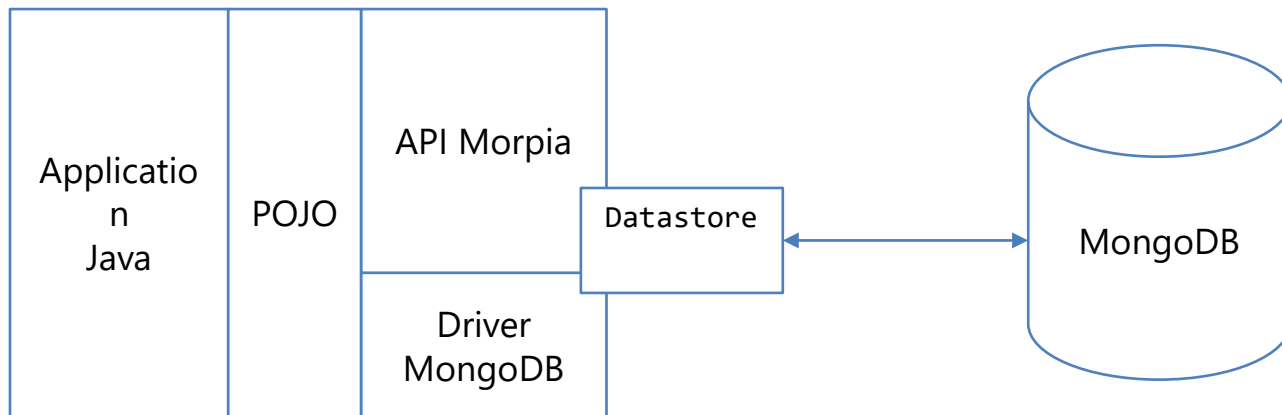
Module 12

API Morphia



Architecture

- MongoDB propose une API de persistance de type JPA (ORM)
 - Mapping des classes Java (POJO) et des documentsDBObject
- Création d'une base Datastore



Installation

- Ajout de la dépendance morphia

```
<dependencies>
  <dependency>
    <groupId>org.mongodb.morphia</groupId>
    <artifactId>morphia</artifactId>
    <version>1.3.1</version>
  </dependency>
</dependencies>
```

Connexion à une base de donnée

- Connexion avec le driver Java MongoClient
- Création d'une base Datastore

```
Morphia morphia = new Morphia();  
Datastore datastore =  
    morphia.createDatastore(new MongoClient(), "banque-morphia");
```

Entités persistantes

- Un objet persistant doit être une instance d'une simple classe Java (POJO).
- Cette classe doit respecter le standard des javabeans:
 - Propriétés privé
 - Getters et setters
 - Constructeur par défaut (sans paramètre)
- Les types utilisables pour les propriétés sont :
 - les types primitifs
 - les classes String et LocalDate
- Le mapping est défini par annotation

Annotations

@Entity	Classe persistante associée à une collection MongoDB
@Id	Attribut identifiant
@Property	Attribut persistant (annotation optionnelle)
@Transient	Attribut non persistant
@Embedded	Encapsulation d'une autre entité (document)
@Reference	Référence à une autre entité (identifiant)

Entité persistante

```
@Entity("clients")
public class Client {

    @Id private ObjectId id;
    private String nom;
    private String prenom;
    private LocalDate naissance;

    @Embedded
    private List<Compte> comptes = new LinkedList<>();

    public Client() {
    }

}
```


Création d'un objet

- Mise à disposition de méthodes de creation, modification et suppression d'objets
- La méthode `save()` crée un nouvel objet ou modifie un objet existant

```
Client cli1 = new Client("Leblanc", "Marc");  
datastore.save(cpt1);  
  
cli1.setNom("Jean-Marc");  
datastore.save(cpt1);
```

Parcourir une collection

- La classe Query permet d'exprimer une requête en indiquant l'entité cible (collection)

```
final Query<Client> query = datastore.createQuery(Client.class);

final List<Client> clients = query
    .field("nom").startsWith("L")
    .asList();

for (Client client : clients) {
    System.out.println(client);
}
```



Fin de la formation **MongoDB**

Pour aller plus loin

- ENI Service sur Internet

- Consultez notre site web www.eni-service.fr
 - > Les actualités
 - > Les plans de cours de notre catalogue
 - > Les filières thématiques et certifications
- Abonnez-vous à nos newsletters pour rester informé sur nos nouvelles formations et nos événements en fonction de vos centres d'intérêts.
- Suivez-nous sur les réseaux sociaux
 -  > Twitter : <http://twitter.com/eniservice>
 -  > Viadeo : <http://bit.ly/eni-service-viadeo>

Pour aller plus loin

- Notre accompagnement

- Tous nos Formateurs sont également Consultants et peuvent :
 - > Vous accompagner à l'issue d'une formation sur le démarrage d'un projet.
 - > Réaliser un audit de votre système d'information.
 - > Vous conseiller, lors de vos phases de réflexion, de migration informatique.
 - > Vous guider dans votre veille technologique.
 - > Vous assister dans l'intégration d'un logiciel.
 - > Réaliser complètement ou partiellement vos projets en assurant un transfert de compétence.

Votre avis nous intéresse

Nous espérons que vous êtes satisfait de votre formation.

Merci de prendre quelques instants pour nous faire un retour en remplissant le questionnaire de satisfaction.

Merci pour votre attention,
et au plaisir de vous revoir prochainement.

