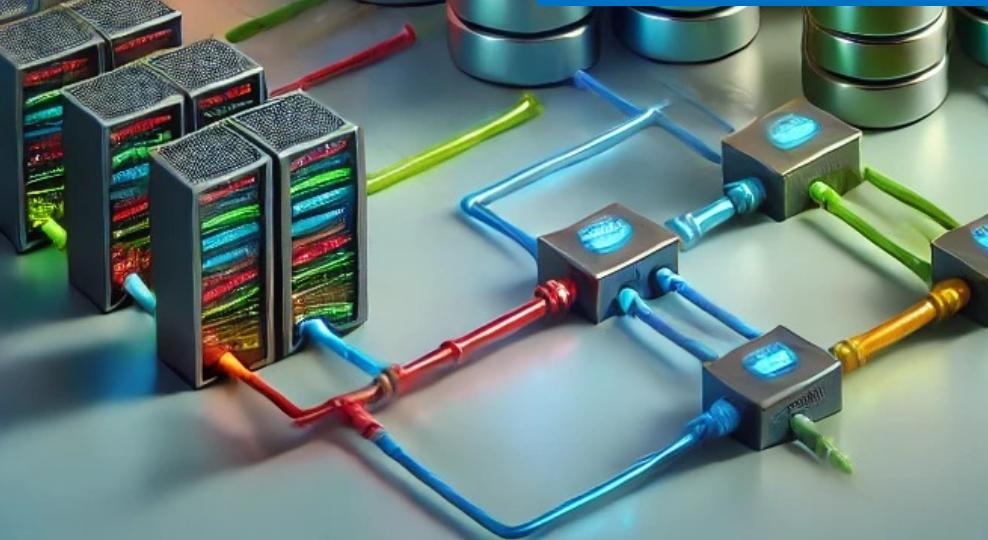




**UNIVERSITE DE CORSE**  
**Master informatique 1<sup>ère</sup>**  
**année**  
**Parcours DFS et DE**  
**2025-2026**



**Bases de données NoSQL**  
**CH3 – MongoDB**



Evelyne VITTORI  
[vittori\\_e@univ-corse.fr](mailto:vittori_e@univ-corse.fr)

# Plan du cours

CH1 – BD  
Relationnelles  
(Compléments)

- Anomalies de mise à jour
- Normalisation-dénormalisation

CH2 – BD NoSQL

- Principes des BD NoSQL

CH3 - MongoDB

- Initiation à MongoDB

# Intoduction à MongoDB

- Installation
- Manipulation en JavaScript
  - Connexion à une BD
  - Insertion de documents
  - Recherche de documents
  - Mises à jour
  - Manipulations globales
- Drivers et langages hôtes
- Conception d'une BD



# Mongo DB

- SGBD noSQL de type documentaire
- Basé sur le format de données JSON
- Développé en C++
- Langage de requête spécifique
- Pas de transactionnel
- Adapté à une organisation distribuée des traitements et des données

Dans ce cours nous ne présenterons que le mode d'utilisation centralisé

# Base de données MongoDB

- BD MongoDB = ensemble de **collections**
- Collection = ensemble de **documents**

	COLLECTION MongoDB	TABLE RELATIONNELLE
Structure	Aucune déclaration préalable Pas de structure figée	Structure figée (schéma) devant être déclarée
Champs	types simples, <b>tableaux</b> , <b>objets</b> , autres documents	types simples
Eléments	<b>documents</b> pouvant avoir des structures différentes	<b>tuples</b> ayant une structure commune

# Notion de document MongoDB

- Document = **objet JSON**
  - Un document est stocké au format BSON (Binary JSON = extension du format **JSON** avec support de types supplémentaires).
- Taille limitée à **16 Mo**
- Chaque document dispose d'une **clé unique** générée automatiquement et permettant de l'identifier dans la collection.

# Le format JSON

## JavaScript Object Notation

- Format textuel permettant de représenter des objets structurés
- Inspiré de la représentation des objets en JavaScript
- Crée par Douglas Crockford en 2006

### Avantages

compréhensible  
par tous

indépendant des  
langages de  
programmation

regroupe des  
données de types  
différents

plus facile à parser  
que XML

# Le format JSON

- Ensemble de paires clé/valeur

```
{ nom: "Rousseau",
```

valeur d'un type de base

```
    age: 30,
```

```
    adresse:
```

```
        { cp : "60950",  
          ville : "Ermenonville"},
```

objet JSON

```
matieres_preferees :
```

```
    ["litterature", "philosophie"],
```

```
}
```



tableau

# Validation d'un document json

- Comment savoir si un document JSON est syntaxiquement correct?
- Validateur en ligne:
  - Un exemple: <https://jsonlint.com/>
  - Il suffit de copier/coller votre document dans le validateur pour vérifier s'il est bien formé (ou non...)



# Se préparer à MongoDB

Installation, démarrage ....

# Fonctionnement de MongoDB

- MongoDB fonctionne en mode classique **client/serveur**.
- Le serveur **mongod** est par défaut en attente sur le port 27017
- Modes d'interaction avec MongoDB:
  - Clients graphiques
  - Shell interactif mongo ou **mongosh** (JavaScript)
  - Drivers pour langage Hôte

nouveau shell sur  
mongodb version 5.

# Installer MongoDB avec Docker

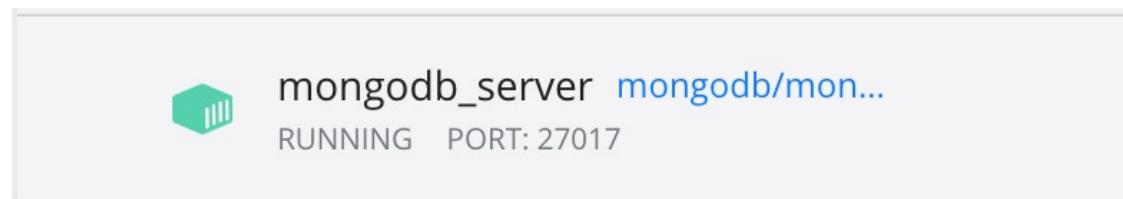
## 1. Lancement de MongoDB Community Server

Ouvrez un terminal et exécutez la commande suivante pour télécharger l'image de MongoDB Community Server et démarrer un conteneur :

```
docker run --name mongodb_community -p 27017:27017 -d mongo:latest
```

## 2. Vérification de l'installation

Assurez-vous que le conteneur est en cours d'exécution.



# Autre solution

## Sous Macos: Installer avec Homebrew

Homebrew est un gestionnaire de packages qui permet d'installer facilement des applications et des logiciels sous macOS

- Ouvrez une fenêtre terminal
- Installez home brew en saisissant la commande suivante

```
/bin/bash -c "$(curl -fSSL  
https://raw.githubusercontent.com/Homebrew/install/HEAD/in  
stall.sh)"
```

- Télécharger l'installation mongodb pour brew

```
brew tap mongodb/brew
```

- Lancer l'installation

```
brew install mongodb-community@7.0
```

# Autre solution : Lancer le serveur (sous MacOs avec homebrew)

- lancer MongoDB en tant que service

*brew services start mongodb-community@7.0*

*ou restart pour le relancer*

- pour vérifier que le service tourne :

*brew services list*

```
Name          Status  User      File
mongodb-community  started vittori_e  ~/Library/LaunchAgents/homebrew.mxcl.mongodb-community.plist
```

Le serveur se place en écoute  
sur le port 27017

# Lancer le serveur manuellement

Inutile si vous utilisez un conteneur Docker : le serveur est lancé automatiquement

- Ouvrir une fenêtre d'invite de commandes  
commande *mongod --dbpath c:\mongodb\data\db*

```
mongod --dbpath c:\mongodb\data\db
```

chemin vers votre répertoire db

- Si tout se passe bien :

```
[initandlisten] waiting for connections on port 27017
```

Le serveur se place en écoute sur le port 27017

# Interagir avec Mongo DB?



Plusieurs clients potentiels!

- Interpréteur de commande *mongoDB interactive shell* + Scripts JavaScript
  - Shell mongo (déprécié)
  - Shell Mongosh (version 5 MongoDB),  
Notre choix pour nous initier à mongoDB
- Clients graphiques (*équivalents à phpMyAdmin pour MySQL*)
  - MongoDB Compass
  - RoboMongo <http://robomongo.org/download.html>  
Notre choix
  - RockMongo
  - ExtensionMongoDb dans VSCode  
Notre choix
- Programmation au sein d'un langage hôte à

# Exemple de client graphique: Client RoboMongo

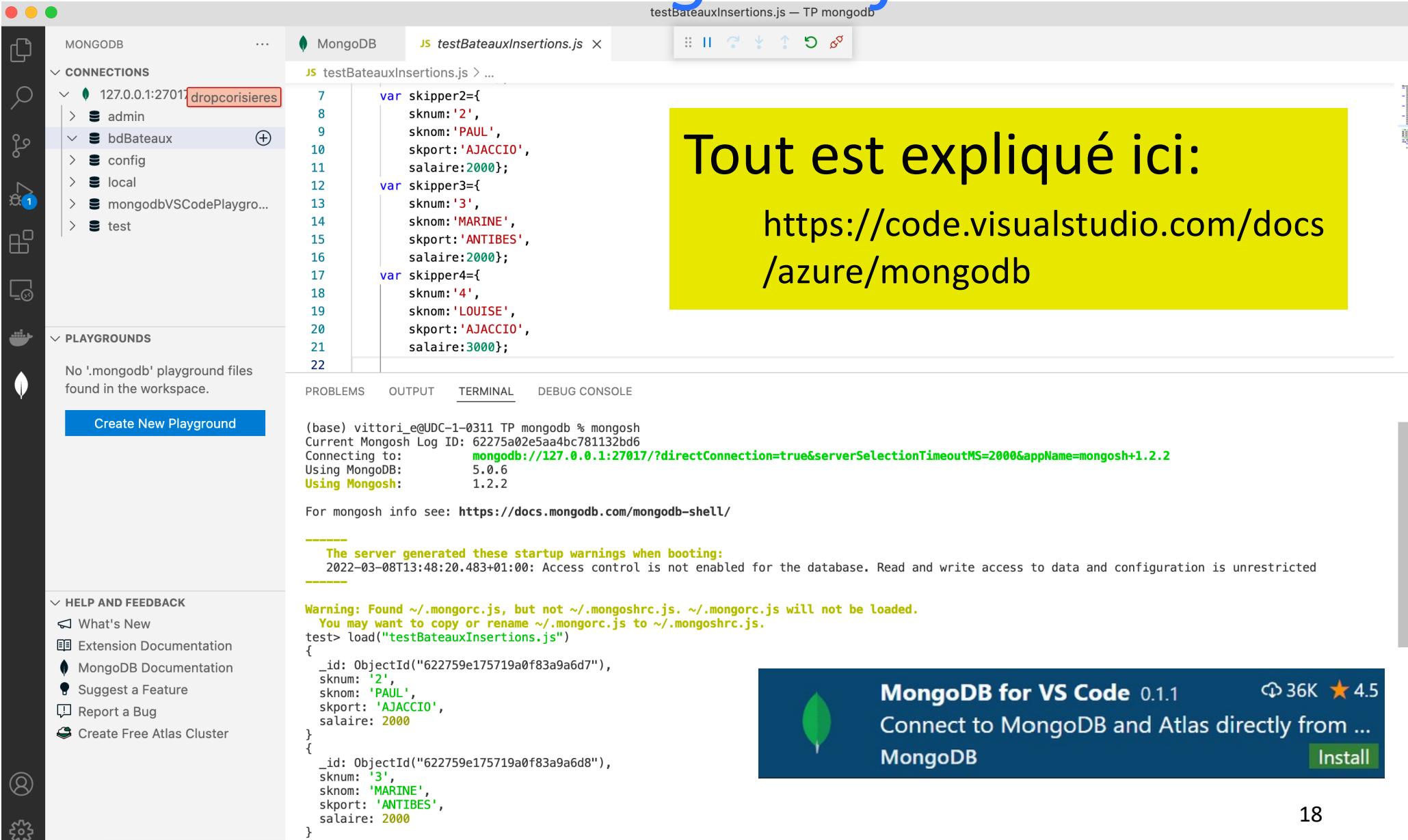
**Visualisation  
d'une  
collection**

**Bases de  
données**

The screenshot shows the Robomongo application window. On the left, the sidebar displays a tree view of database connections and collections. A red box highlights the 'System' and 'Collections' sections under a connection named 'New Connection (8)'. A blue box highlights the 'etudiants' collection under the 'Collections' section. A red arrow points from the text 'Bases de données' to the highlighted sidebar area. A blue arrow points from the text 'Visualisation d'une collection' to the highlighted collection table. The main panel shows a query bar with 'db.getCollection('etudiants').find({})' and a results table titled 'etudiants' with 0 seconds execution time. The table has columns 'Key', 'Value', and 'Type'. The data is presented in an expandable list format, showing three documents. Document (1) has fields \_id, nom, and prenom with values ObjectId("56589e5d091ad2d10c8a8cd..."), Paoli, and Pascal respectively. Document (2) has fields \_id, nom, titre, and age with values ObjectId("5658a09b091ad2d10c8a8cd..."), Bonaparte, empereur, and 20 respectively. Document (3) has fields \_id, nom, prenom, age, and matieres\_preferees with values ObjectId("5658b1d3091ad2d10c8a8cd..."), Rousseau, Jean-Jacques, 30.000000, and an array [philosophie, litterature] respectively.

Key	Type
(1) ObjectId("56589e5d091ad2d10c8a8cd...")	Object
_id	ObjectId
nom	String
prenom	String
(2) ObjectId("5658a09b091ad2d10c8a8cd...")	Object
_id	ObjectId
nom	String
titre	String
age	Int32
(3) ObjectId("5658b1d3091ad2d10c8a8cd...")	Object
_id	ObjectId
nom	String
prenom	String
age	Double
matieres_preferees	Array
0	String
1	String

# Une solution confortable : *Extension MongoDB for VScode*



The screenshot shows the MongoDB extension interface in Visual Studio Code. On the left, the sidebar displays 'CONNECTIONS' with a connection to '127.0.0.1:27017' named 'dropcorisieres'. Below it is a 'PLAYGROUNDS' section indicating no files found. A 'Create New Playground' button is present. On the right, the main area shows a code editor with a file named 'testBateauxInsertions.js' containing the following code:

```
var skipper2={  
    sknum:'2',  
    sknom:'PAUL',  
    skport:'AJACCIO',  
    salaire:2000};  
  
var skipper3={  
    sknum:'3',  
    sknom:'MARINE',  
    skport:'ANTIBES',  
    salaire:2000};  
  
var skipper4={  
    sknum:'4',  
    sknom:'LOUISE',  
    skport:'AJACCIO',  
    salaire:3000};
```

Below the code editor is a terminal window showing the output of running the script:

```
(base) vittori_e@UDC-1-0311 TP mongodb % mongosh  
Current MongoDB Log ID: 62275a02e5aa4bc781132bd6  
Connecting to: mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.2.2  
Using MongoDB: 5.0.6  
Using Mongosh: 1.2.2  
  
For mongosh info see: https://docs.mongodb.com/mongodb-shell/  
  
----  
The server generated these startup warnings when booting:  
2022-03-08T13:48:20.483+01:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted  
  
----  
  
Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be loaded.  
You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.  
test> load("testBateauxInsertions.js")  
{  
    _id: ObjectId("622759e175719a0f83a9a6d7"),  
    sknum: '2',  
    sknom: 'PAUL',  
    skport: 'AJACCIO',  
    salaire: 2000  
}  
  
{  
    _id: ObjectId("622759e175719a0f83a9a6d8"),  
    sknum: '3',  
    sknom: 'MARINE',  
    skport: 'ANTIBES',  
    salaire: 2000  
}
```

A yellow callout box on the right side of the terminal area contains the text 'Tout est expliqué ici:' followed by a link: <https://code.visualstudio.com/docs/azure/mongodb>.

**MongoDB for VS Code** 0.1.1      36K      4.5  
Connect to MongoDB and Atlas directly from ...  
MongoDB      Install

# Extension MongoDB for VScode

## Playground mongodb

The screenshot shows the MongoDB extension interface in VSCode. On the left, the sidebar displays connections to 'localhost:27017' and a playground named 'MongoDB Playground.mongodb'. The main area is a code editor titled 'playground.mongoDB.js' containing the following MongoDB query:

```
1 // Select the database to use.
2 use('bdBateaux');
3 db=connect("bdBateaux");
4 db.skippers.find().pretty();
5
```

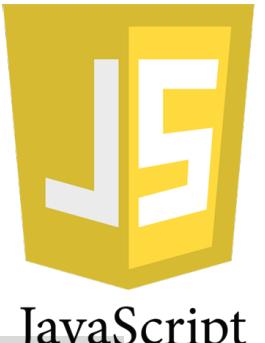
A green arrow points from the text 'playground.mongoDB.js' to the code editor. Another green arrow points from the text 'fenêtre d'affichage du résultat' to the results panel on the right. The results panel is titled 'Playground Result' and shows the output of the query:

```
1 {
2   "_id": {
3     "$oid": "67345b06d675b8f347cc24"
4   },
5   "sknum": 10,
6   "sknom": "JEAN",
7   "skport": "AJACCIO",
8   "salaire": 3000
9 },
10 {
11   "_id": {
12     "$oid": "67345b06d675b8f347cc24"
13   },
14   "sknum": 20,
15   "sknom": "PAUL",
16   "skport": "AJACCIO",
17   "salaire": 2000
18 }
```

Le plus confortable pour tester :  
Exécution interactive de requêtes

# Shell mongosh + scripts JavaScript

- Créer un fichier JavaScript dans un éditeur



testMongoDB.js

```
var dbs=db.adminCommand("listDatabases");
    //retourne un objet contenant la
liste des bases de données
printjson(dbs);
```

Autre solution  
pour tester!

# Shell mongosh + scripts JavaScript

- Pour Exécuter un fichier js, il faut :

- lancer le shell mongosh

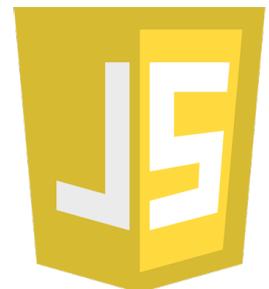
```
mongosh
```

```
....
```

```
[test>
```

- utiliser la fonction load

```
[test> load ("testMongoDB.js")
```



JavaScript

# Documentation MongoDB

- Documentation officielle

<https://docs.mongodb.org/manual/>

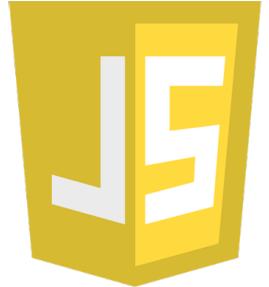
- Opérations CRUD
- Tutoriaux





## Connexion à une BD

# Connexion à une BD



JavaScript

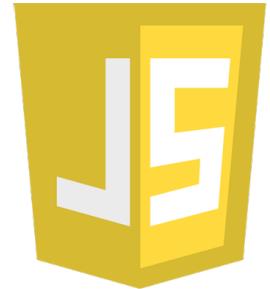
- Création d'une base de données ou connexion à une BD existante

```
db=connect("nomBD");
```



Si la BD n'existe pas, elle sera créée

# Connexion à une BD



JavaScript

- Dans un terminal, après avoir saisi la commande mongosh

ou directement dans un playground

```
test> db=connect("bdBateaux");
bdbateaux
bdbateaux> dbs=db.adminCommand("listDatabases");
{
  databases: [
    { name: 'admin', sizeOnDisk: Long('40960'), empty: false },
    { name: 'bdbateaux', sizeOnDisk: Long('40960'), empty: false },
    { name: 'config', sizeOnDisk: Long('110592'), empty: false },
    { name: 'local', sizeOnDisk: Long('40960'), empty: false }
  ],
  totalSize: Long('233472'),
  totalSizeMb: Long('0'),
  ok: 1
}
```

Retourne un objet JSON  
contenant la liste des bases  
de données



# mongoDB

## Insertion de documents

# Insérer des documents dans une collection

Collection Etudiants

Clés

	56589e5d091ad	5658a09b091ad
--	---------------	---------------

Valeurs

nom: "Pascal"  
prenom: "Paoli"

nom: "Bonaparte"  
prenom: "Napoleon"  
titre: "Empereur"

\_id clé unique générée  
par MongoDB

# Insérer des documents dans une collection

Shell  
Mongosh

- Deux commandes :
  - db.**collection.insertOne**(objet JSON)
    - Insertion d'un seul document
  - ou
  - db.**collection.insertMany**(tableau d'objets JSON)
    - Insertion de plusieurs documents transmis dans un tableau

si la collection n'existe pas,  
elle sera créée!!

# Insérer des documents dans une collection



```
db=connect ("bdEtudiants") ;  
  
var etudiant1={  
    nom: "Paoli",  
    prenom: "Pascal"} ;  
  
var etudiant2={  
    nom: "Bonaparte",  
    prenom: "Napoleon",  
    titre: "Empereur"} ;  
  
db.etudiants.insertOne(etudiant1) ;  
db.etudiants.insertOne(etudiant2) ;
```

déclaration d'objets  
JavaScript



# mongoDB

## Recherche de documents

# Rechercher des documents

- Langage de requête spécifique
- Principe: Recherche « par motif » (pattern)
  - le critère de recherche est représenté par un objet JSON (le « pattern » ou « motif »)

malgré tout  
équivalent à une  
clause Where en  
SQL



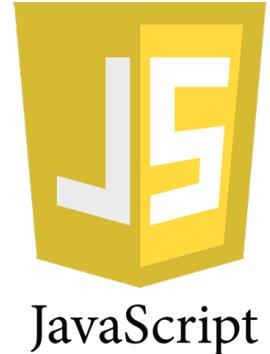
# Rechercher des documents

`db.collection.find (query, projection)` : renvoie un tableau de documents (appelé « **curseur** »)

- 2 paramètres optionnels
  - **query**: objet servant de critère de recherche
  - **projection**: indique les attributs que l'on souhaite retourner
    - `{nom: 1}` indique que seul l'attribut *nom* sera retourné
    - `{nom:0}` indique que tous les attributs seront retournés sauf *nom*

-si *projection* n'est pas précisé, tous les attributs sont retournés  
- l'attribut `_id` est toujours retourné par défaut

# Rechercher tous les documents d'une collection



- Visualiser tous les documents d'une collection

`db.collection.find()`

objet de type Cursor

```
db=connect ("bdEtudiants");
var etud=db.etudiants.find ();
```

**etud.forEach** (

```
function (etu) {
    printjson (etu);
} );
```

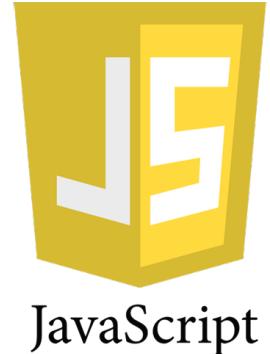
Affichage des documents du tableau résultat

Autres écritures possibles:

```
...find(null);
...find(undefined);
...find({});
```

```
connecting to: bdEtudiants
{
    "_id" : ObjectId("5658e564c4cee8a98f5eb0c4"),
    "nom" : "Paoli",
    "prenom" : "Pascal"
}
{
    "_id" : ObjectId("5658e564c4cee8a98f5eb0c5"),
    "nom" : "Bonaparte",
    "prenom" : "Napoleon",
    "titre" : "Empereur"
}
```

# Rechercher tous les documents d'une collection



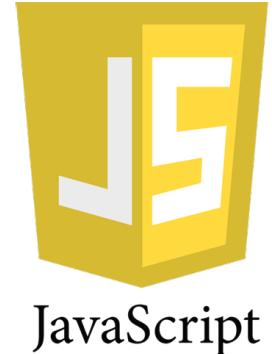
- Visualiser tous les documents d'une collection  
`db.collection.find()`

```
db=connect ("bdEtudiants") ;  
db.etudiants.find() .forEach (printjson) ;
```



Une solution plus simple !

# Rechercher tous les documents d'une collection



- Encore plus simple à partir de playground

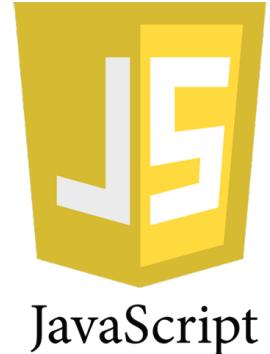
A screenshot of the MongoDB playground interface. On the left, the code editor shows a single-line command: `db=connect("bdEtudiants"); db.etudiants.find();`. The output window on the right displays the results of the query, which are two documents represented as JSON objects. A large yellow callout box with the text "Affichage des documents du tableau résultat" points to the result list. A blue arrow points from the text "Affichage des documents du tableau résultat" to the first document in the result list.

```
JS db=connect("bdEtudiants"); /Users/vittori_ ⏎ ... { } Playground Result ...
```

```
1 Currently connected to localhost:27017. Click here to c
2 db=connect("bdEtudiants");
3 db.etudiants.find();
4
5 [
6   {
7     "_id": {
8       "$oid": "6617970db867d35b81a461"
9     },
10    "nom": "Paoli",
11    "prenom": "Pascal"
12  },
13  {
14    "_id": {
15      "$oid": "6617970db867d35b81a461"
16    },
17    "nom": "Bonaparte",
18    "prenom": "Napoleon",
19    "titre": "Empereur"
20  }
21 ]
```

Affichage des documents du tableau résultat

# Rechercher tous les documents d'une collection



- Visualiser uniquement certains champs

```
db=connect ("bdEtudiants");
var etud=db.etudiants.find( {}, { _id:0, nom:1 });

etud.forEach (
  function(etu) {
    printjson(etu);
  } );

```

connecting to: bdEtudiants  
{ "nom" : "Obama" }  
{ "nom" : "Paoli" }  
{ "nom" : "Rousseau" }

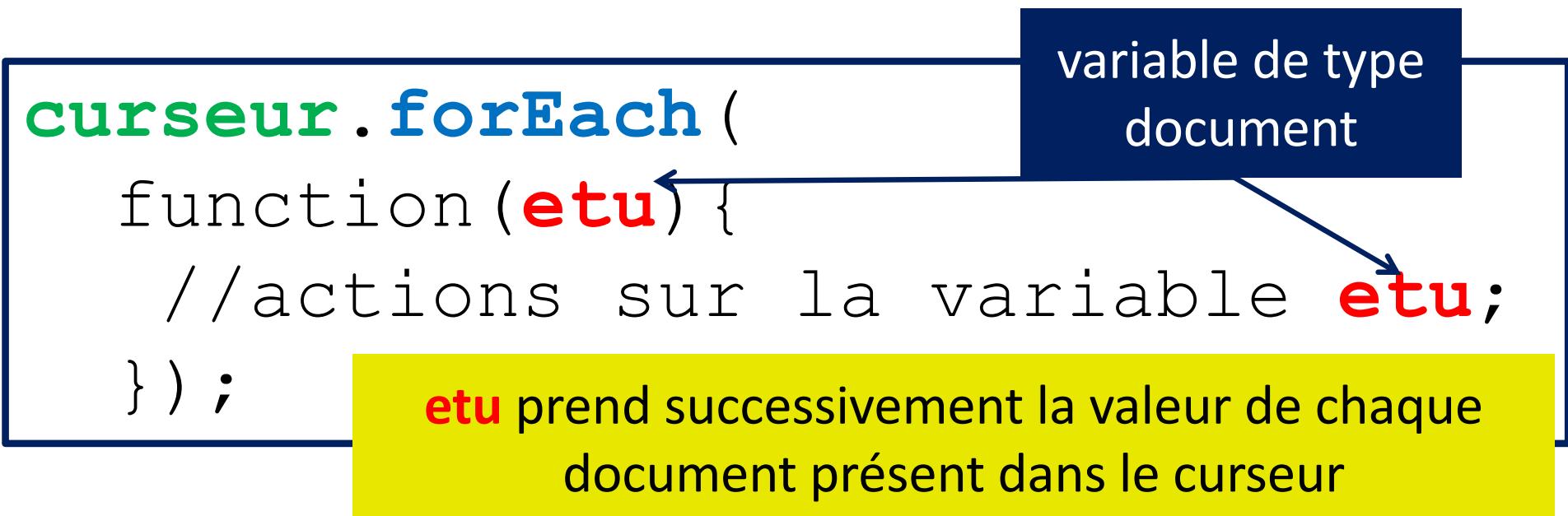
```
db.etudiants.find( {}, { _id:0, nom:1 } ).  
  forEach(printjson);
```

Une solution plus simple !

# La méthode `foreach`

**forEach** est une méthode de la classe Cursor

- soit c un objet de type Cursor
  - `c.forEach(<function>)`
- le paramètre est une fonction applicable sur chacun des documents du curseur

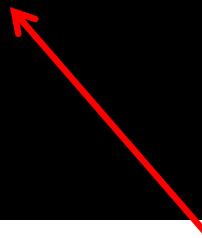


# Rechercher des documents selon des critères de sélection

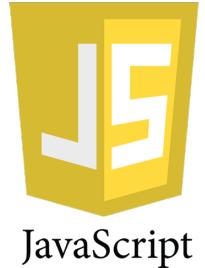


```
var etud=db.etudiants.find(  
    {nom: "Bonaparte"} );  
etud.forEach(  
    function(etu) {  
        printjson(etu);  
    } );
```

```
connecting to: bdEtudiants  
{  
    "_id" : ObjectId("5658e564c4cee8a98f5eb0c5"),  
    "nom" : "Bonaparte",  
    "prenom" : "Napoleon",  
    "titre" : "Empereur"  
}
```



Le champ `_id` est généré automatiquement lors de l'insertion des documents



# Version avec playground

The screenshot shows a MongoDB playground interface. On the left, the code pane contains the following JavaScript-like code:

```
1 db=connect("bdEtudiants");
2 db.etudiants.find({nom: "Bonaparte"});
3
```

On the right, the results pane displays the response:

```
1 [
2   {
3     "_id": {
4       "$oid": "6617970db867d35b81a46150"
5     },
6     "nom": "Bonaparte",
7     "prenom": "Napoleon",
8     "titre": "Empereur"
9   }
10 ]
```

Le champ `_id` est généré automatiquement lors de l'insertion des documents

# A vous de jouer (1-2) ..



- Connectez-vous à mongoDB et créez une base de données bdBateaux
1. Créez une collection mongoDB « skippers » et y insérer un document pour chacun des skippers décrits dans le tableau suivant :

SKNUM	SKNOM	SKPORT	SALAIRE
1	JEAN	AJACCIO	3000
2	PAUL	AJACCIO	2000
3	PIERRE	ANTIBES	aucun
4	MARIE	BASTIA	1500

## 2. Affichages

- Afficher tous les documents de la collection skippers
- Afficher la liste des noms et ports des skippers présents dans la collection skippers

# Rechercher des documents avec des critères de comparaison

- Opérateurs de comparaison

- **\$gt** : plus grand que
- **\$lt** : plus petit que
- **\$gte** : plus grand ou égal à
- **\$lte**: plus petit ou égal à

Utilisés comme des champs dans un document json

```
//Rechercher les étudiants ayant au moins 20 ans
```

```
db.etudiants.find( { age: { $gte:20 } } );
```

# Rechercher des documents avec des connecteurs logiques

Les conditions composées par des connecteurs logiques AND (**\$and**) et OR (**\$or**) sont exprimées dans un **tableau** dont chaque élément est un objet exprimant une condition.

```
//Rechercher les étudiants dont le nom  
est « paoli » ou l'age ≥20  
db.etudiants.find(  
{ $or:  
[ {age: {$gte:20 } } ] ,  
{ nom: "Paoli" } ]  
} );
```

Condition 1

Condition 2

# Rechercher selon l'existence d'un champ

```
//Rechercher les étudiants possédant un champ titre
var et=db.etudiants.find(
    {titre :
        { $exists :true }
    } );
```

# Rechercher selon le type d'un champ

//Rechercher les étudiants ayant le champ nom à null ou sous forme de chaîne de caractères

```
var et=db.etudiants.find( {  
    $or : [ { nom : { $type :2 } } ,  
            { nom : { $type :10 } } ] } ) ;
```

Type	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Undefined	6
Object id	7
Boolean	8
Date	9
Null	10

Liste des types:  
<https://docs.mongodb.org/manual/reference/operator/query/type/>

# Rechercher des documents

- Connaitre le **nombre de documents** trouvés

`db.collection.find(...).count()`

countDocument() dans le shell mongosh

méthode d'un  
objet cursor

- Rechercher le premier document satisfaisant une condition

`db.collection.findOne (query) :`

- renvoie le premier document correspondant aux critères de l'objet query
- renvoie null si aucun document n'est trouvé

# Rechercher dans des sous-documents

//Rechercher les étudiants habitant la ville d'Ajaccio

```
var et=db.etudiants.find(  
    { "adresse.ville": "Ajaccio" } );
```

" obligatoires

attribut du sous-document

```
        "_id" : ObjectId("565b4215dfce6d1f3759ceee"),  
        "nom" : "Bonaparte",  
        "prenom" : "Napoleon",  
        "adresse" : {  
            "rue" : "1 rue Saint Charles",  
            "cp" : "20000",  
            "ville" : "Ajaccio"  
        }
```

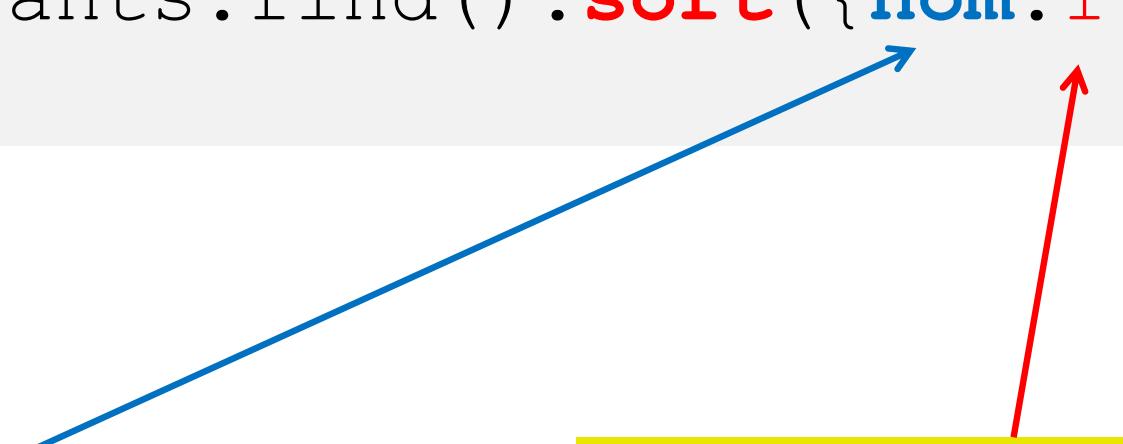
Pour un tableau, la syntaxe est similaire : "champTableau.numero"

# Trier des documents recherchés

```
//resultat trié par ordre croissant de nom  
var etud=  
db.etudiants.find().sort({nom:1});
```

champ  
de tri

1 ordre croissant  
-1 ordre décroissant



# A vous de jouer (3)..



- Connectez-vous à votre base de données bdBateaux
  1. Afficher la liste des noms des skippers de la collection skippers habitant Ajaccio et ayant un salaire strictement supérieur à 2000 euros.
  2. Afficher les numéros et noms des skippers qui n'ont pas de salaire.



# Mises à jour de documents

Modifications,  
Suppression

# Commandes de mises à jour

Shell  
Mongosh

- db.collection.**updateOne** (**query**, **update**) :  
modification d'un seul document
- db.collection.**updateMany** (**query**, **update**) :  
modification de plusieurs documents

# Modifier des documents

- db.collection.**updateOne** (**document**) : sauvegarde le document dans la collection en effectuant une insertion ou une mise à jour.

```
var etud=db.etudiants.findOne(  
    { nom: "Bonaparte" }) ;  
if (etud) {  
    etud.titre = "premier consul";  
    db.etudiants.updateOne(  
        { _id: etud._id },  
        { $set: { titre: etud.titre } } );  
} else {  
    print("Étudiant introuvable !");  
}
```

# Ajouter/Supprimer un attribut d'un document

```
db.etudiants.updateOne ( critère de recherche  
  { nom: "Bonaparte" } ,  
  { ajout ou mise à jour  
    $set: { age: 30 } , du champ age  
    $unset: { titre: "" }  
  } ) ; } suppression du  
                                champ titre
```

# Modifier des documents

```
db.etudiants.updateOne(  
  {nom:"Bonaparte"},  
  {$set : {age: 30,  
           titre : "empereur"} } );
```

ajout ou mise à jour des champs des documents sélectionnés

```
db.etudiants.updateOne(  
  {nom:"Bonaparte"},  
  {$inc : {age: 1}} );
```

incrémentation du champ age

```
{  
  "_id" : ObjectId("565b4215dfce6d1f3759ceee"),  
  "nom" : "Bonaparte",  
  "prenom" : "Napoleon",  
  "adresse" : {  
    "rue" : "1 rue Saint Charles",  
    "cp" : "20000",  
    "ville" : "Ajaccio"  
  },  
  "age" : 31,  
  "titre" : "empereur"
```

# Modifier plusieurs documents

```
db.etudiants.updateMany (  
    {titre : { $exists :false} } ,  
    {$set : {titre: null} }  
);
```

sélection des documents où le champ titre n'existe pas

ajout d'un champ titre vide (=null)

# Modifier des sous-documents

```
db.etudiants.updateOne (           modification du champ  
{nom: "Bonaparte"},                ville de l'objet interne  
{$set : {"adresse.ville": "Paris"} } );      adresse
```

```
{  
    "_id" : ObjectId("565b4215dfce6d1f3759ceee"),  
    "nom" : "Bonaparte",  
    "prenom" : "Napoleon",  
    "adresse" : {  
        "rue" : "1 rue Saint Charles",  
        "cp" : "20000"  
        "ville" : "Paris"  
    },  
    "age" : 31,  
    "titre" : "empereur"  
}
```

# Modifier des sous-documents

```
db.etudiants.updateOne (  
  { nom: "Rousseau" } ,  
  { $push : {  
    matieresPreferees : "Politique" } }  
);
```

ajout d'un élément au tableau

```
{  
  "_id" : ObjectId("5666fc15575ff6cf51badbb6"),  
  "nom" : "Rousseau",  
  "age" : 30,  
  "adresse" : {  
    "cp" : "60950",  
    "ville" : "Ermenonville"  
  },  
  "matieresPreferees" : [  
    "litterature",  
    "philosophie",  
    "Politique"  
  ]  
}
```

# Commandes de suppression

- db.collection.**deleteOne** (**query**) :  
suppression du premier document vérifiant le critère spécifié dans **query**

```
db.etudiants.deleteOne( { nom: "Bonaparte" } )
```

- db.collection.**deleteMany** (**query**) :  
suppression de tous les documents vérifiant le critère spécifié dans **query**

```
db.etudiants.deleteMany( { age: { $lt: 20 } } );
```

# A vous de jouer (4)..



- Définissez les commandes permettant de réaliser les actions suivantes:
  1. ajouter un champ salaire =1500 euros à tous les skippers qui n'ont pas de salaire
  2. augmenter de 500 euros le salaire de tous les skippers
  3. ajouter un champ specialites=« barreur, transatlantique » au skipper de nom « Jean »



# mongoDB

## Manipulations globales

Collections, Bases de données

# Manipulation des collections

- `db.collection.drop()` : supprime une collection
- `db.collection.renameCollection(`

*nouveauNom,*

{ `dropTarget: true` } );

facultatif

Renomme une collection

Si une collection avec le même nom existe déjà, une erreur sera générée à moins que l'option `dropTarget` soit définie à `true`.

- `db.getName(): String` : renvoie le nom de la BD active

# Manipulation des BD

- db.**getCollectionNames()** : String[] : renvoie un tableau contenant les noms des collections de la BD active
- db.**dropDatabase()**: supprime la BD active
- db.**dropDatabase()**: supprime la BD active
- **mongodump** et **mongorestore** : sauvegarde (restore) une BD mongo

```
mongodump --uri "mongodb://localhost:27017" --db  
myDatabase --out /backup/mongodb
```



# mongoDB

## Index et Optimisation

# Définition d'index

- Comme dans les SGBDR, la définition d'index:
  - permet d'améliorer les performances lors des recherches
  - Mais pénalise les mises à jour ...
- Un **index** peut être défini sur un ou plusieurs champs
- Possibilité de définir des index uniques

cf. clés primaires

Le fonctionnement des index est très similaire au monde SQL

# Définition d'index

db.collection.**createIndex** ({champ : **type**, .. } ,  
**options**) : crée un index sur le ou les champs spécifiés

- **type**: -1 (descendant) ou 1 (ascendant)
- **options**:
  - {**unique**:true}: crée un index vérifiant l'unicité

```
db.etudiants.createIndex( { nom: 1 } ) ;  
db.etudiants.createIndex( { numEtud: 1 } ,  
{ unique: true } )
```

L'index ne sera pas créé si les valeurs existantes du champ ne sont pas uniques

# Affichage de la liste des index

```
db.etudiants.createIndex( { nom: 1 } )  
printjson(db.etudiants.getIndexes());
```

Index par défaut  
sur le champ \_id

```
[{"v": 1, "key": {"_id": 1}, "name": "_id_", "ns": "bdEtudiants.etudiants"}, {"v": 1, "key": {"nom": 1}, "name": "nom_1", "ns": "bdEtudiants.etudiants"}]
```

# Comment savoir si un index est utilisé ?

- En visualisant le **plan d'exécution** d'une requête.

Currently connected to localhost:27017. Click here to change connection.

```
1 db=connect("bdEtudiants");
2 db.etudiants.find({"nom": "Paoli"}).explain();
3
```

{} Playground Result X

```
3   "queryPlanner": {
4     "parsedQuery": {
5       "$query": {
6         "nom": "Paoli"
7       }
8     },
9     "plannedQuery": {
10       "queryHash": "F1BD6F15",
11       "planCacheKey": "A20E3997",
12       "maxIndexedOrSolutionsReached": false,
13       "maxIndexedAndSolutionsReached": false,
14       "maxScansToExplodeReached": false,
15       "winningPlan": {
16         "stage": "FETCH",
17         "inputStage": {
18           "stage": "IXSCAN",
19           "keyPattern": {
20             "nom": 1
21           },
22           "indexName": "nom_1",
23           "isMultiKey": false,
24           "multiKeyPaths": [
25             {
26               "nom": []
27             }
28           ],
29           "isUnique": false,
30           "isSparse": false,
31           "isPartial": false,
32           "indexVersion": 2,
33           "direction": "forward",
34           "scanType": "range"
35         }
36       }
37     }
38   }
39 }
```

Parcours de l'index



# Agregats

# Notion d'Agrégats

- L'agrégation dans MongoDB permet
  - de traiter des données et d'obtenir des résultats calculés en plusieurs étapes.
- Pipeline d'agrégation
- Avantages
  - **Optimisé** pour retourner des résultats rapidement.
  - **Flexibilité** : Peut effectuer des transformations complexes en combinant différentes étapes.
  - **Scalabilité** : Adapté aux grandes bases de données distribuées.



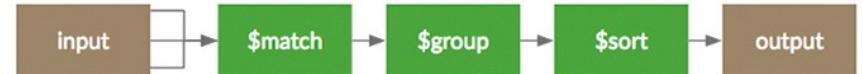
similaire au group  
by SQL

# Etapes d'un pipeline d'agrégation



- **\$match :**
  - Filtre les documents pour ne passer que ceux qui correspondent aux critères spécifiés.
- **\$group :**
  - Regroupe les documents par une *clé spécifiée* et permet d'effectuer diverses opérations sur les données regroupées (somme, moyenne, maximum, minimum, etc.).
- **\$sort :**
  - Trie les documents.
- **\$lookup :**
  - Joint des documents de deux collections en ajoutant un tableau dans la collection initiale.

# Etapes d'un pipeline d'agrégation



- **\$unwind :**
  - "Eclate" les documents comportant un champ tableau en créant un document distinct pour chaque élément du tableau.  
souvent après un lookup
- **\$project :**
  - Définit les champs spécifiques à conserver dans les documents de sortie.
  - Permet aussi d'effectuer des **calculs** ou manipulations sur les champs (par exemple, renommer, créer de nouveaux champs, masquer certains champs, etc.).

Liste non exhaustive

# Exemple 1

- Calculer le salaire total de tous les skippers sauf ceux qui n'ont pas de salaire défini

```
db.skippers.aggregate([
  { $match: { salaire: { $exists: true,
                        $ne: "aucun" } } },
  { $group: {
    _id: null,           // champ de partitionnement ou null
    TotalSalaires: { $sum: "$salaire" }
  }},
  { $project: {
    _id: 0,
    TotalSalaires: 1
  }}
]) .forEach(printjson);
```

champ de partitionnement ou null

opérateur 'not equal' :  
\$ne

ne renvoyer que TotalSalaires

```
[  
 {  
   "TotalSalaires": 6500  
 }  
]
```

# Exemple 1

- Calculer le salaire total de tous les skippers sauf ceux qui n'ont pas de salaire défini

```
{ "SKNUM": "1",
  "SKNOM": "JEAN",
  "SKPORT": "AJACCIO",
  "salaire": 3000 }

{ "SKNUM": "2",
  "SKNOM": "PAUL",
  "SKPORT": "BASTIA",
  "salaire": 2500 }

{ "SKNUM": "3",
  "SKNOM": "LAURA",
  "SKPORT": "ANTIBES",
  "salaire": "aucun" }

{ "SKNUM": "4",
  "SKNOM": "PIERRE",
  "SKPORT": "AJACCIO",
  "salaire": 1500 }
```

\$match  
→

```
{ "SKNUM": "1",
  "SKNOM": "JEAN",
  "SKPORT": "AJACCIO",
  "salaire": 3000 }

{ "SKNUM": "2",
  "SKNOM": "PAUL",
  "SKPORT": "BASTIA",
  "salaire": 2500 }

{ "SKNUM": "4",
  "SKNOM": "PIERRE",
  "SKPORT": "AJACCIO",
  "salaire": 1500 }
```

\$group

```
{ "_id": null,
  "TotalSalaires": 7000 }
```

\$project  
↓

```
{"TotalSalaires": 7000 }
```

# Exemple 2

Trouver le salaire total par port de résidence

```
db.skippers.aggregate([
  {
    $match: { salaire: { $exists: true,
                         $ne: "aucun" } }
  },
  {
    $group: {
      _id: "$skport",
      TotalSalaires: { $sum: "$salaire" }
    }
  }
]);
```

champ de partitionnement ou null

# Exemple 2

Trouver le salaire total par port de résidence

```
{ "SKNUM": "1",
  "SKNOM": "JEAN",
  "SKPORT": "AJACCIO",
  "salaire": 3000 }

{ "SKNUM": "2",
  "SKNOM": "PAUL",
  "SKPORT": "BASTIA",
  "salaire": 2500 }

{ "SKNUM": "3",
  "SKNOM": "LAURA",
  "SKPORT": "ANTIBES",
  "salaire": "aucun" }

{ "SKNUM": "4",
  "SKNOM": "PIERRE",
  "SKPORT": "AJACCIO",
  "salaire": 1500 }
```

\$match



```
{ "SKNUM": "1",
  "SKNOM": "JEAN",
  "SKPORT": "AJACCIO",
  "salaire": 3000 }

{ "SKNUM": "2",
  "SKNOM": "PAUL",
  "SKPORT": "BASTIA",
  "salaire": 2500 }

{ "SKNUM": "4",
  "SKNOM": "PIERRE",
  "SKPORT": "AJACCIO",
  "salaire": 1500 }
```

\$group

```
{ "_id": "AJACCIO",
  "TotalSalaires": 4500
}

{ "_id": "BASTIA",
  "TotalSalaires": 2500
}
```

# Exemple 3

On suppose la création d'une collection croisières

Donner le nombre de croisières par skipper

```
db.skippers.aggregate([
  {
    $lookup: {
      from: "croisières",
      localField: "SKNUM",
      foreignField: "SKNUM",
      as: "croisiere_details"
    }
  },
  {
    $project: {
      SKNOM: 1,
      TotalCroisières: { $size: "$croisiere_details" }
    }
  }
]);
```

ajoute un champ croisiere\_details de type tableau à chaque document de skippers

comptage dans le project

# Exemple 3

Donner le nombre de croisières par skipper

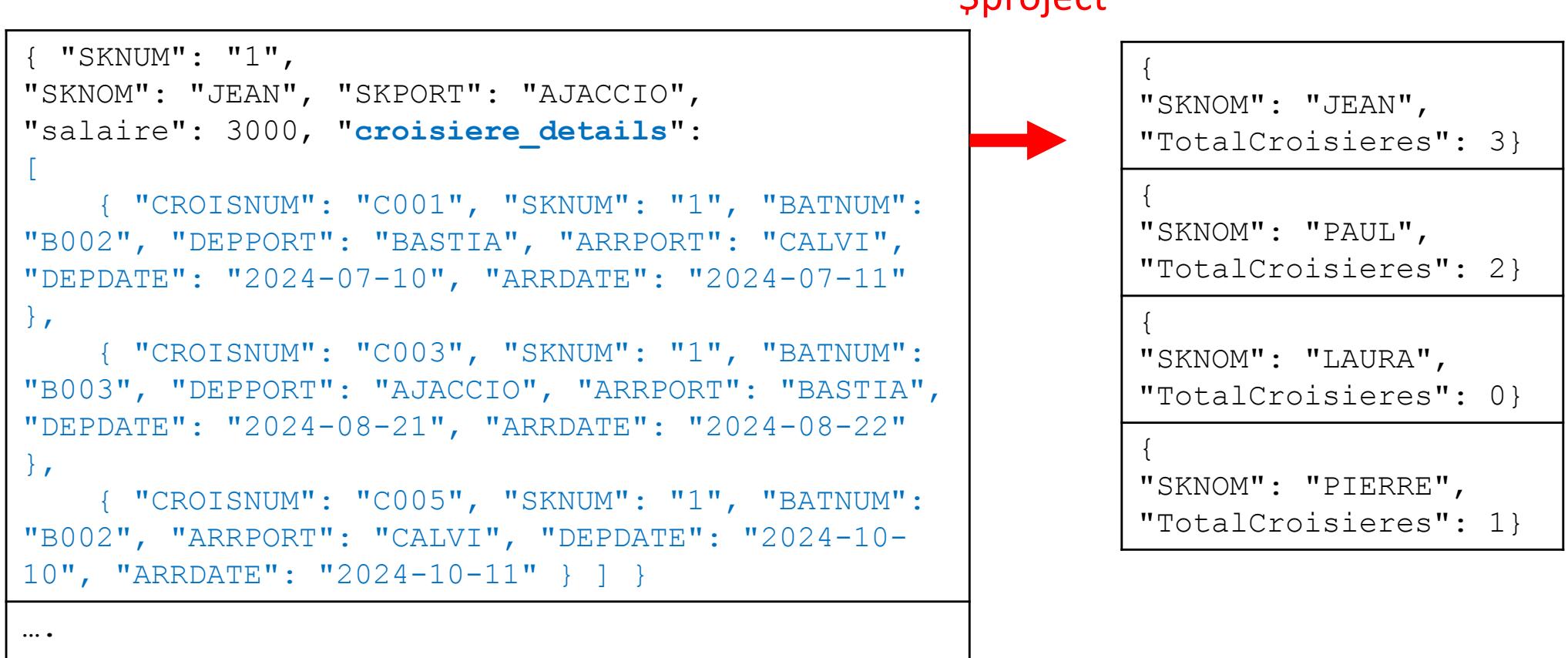
```
{ "SKNUM": "1",  
  "SKNOM": "JEAN",  
  "SKPORT": "AJACCIO",  
  "salaire": 3000 }  
  
{ "SKNUM": "2",  
  "SKNOM": "PAUL",  
  "SKPORT": "BASTIA",  
  "salaire": 2500 }  
  
{ "SKNUM": "3",  
  "SKNOM": "LAURA",  
  "SKPORT": "ANTIBES",  
  "salaire": "aucun" }  
  
{ "SKNUM": "4",  
  "SKNOM": "PIERRE",  
  "SKPORT": "AJACCIO",  
  "salaire": 1500 }
```

\$lookup  
→

```
{ "SKNUM": "1",  
  "SKNOM": "JEAN", "SKPORT": "AJACCIO",  
  "salaire": 3000, "croisiere_details":  
  [  
    { "CROISNUM": "C001", "SKNUM": "1", "BATNUM":  
      "B002", "DEPPORT": "BASTIA", "ARRPORT": "CALVI",  
      "DEPDATE": "2024-07-10", "ARRDATE": "2024-07-11"  
    },  
    { "CROISNUM": "C003", "SKNUM": "1", "BATNUM":  
      "B003", "DEPPORT": "AJACCIO", "ARRPORT": "BASTIA",  
      "DEPDATE": "2024-08-21", "ARRDATE": "2024-08-22"  
    },  
    { "CROISNUM": "C005", "SKNUM": "1", "BATNUM":  
      "B002", "ARRPORT": "CALVI", "DEPDATE": "2024-10-  
      10", "ARRDATE": "2024-10-11" } ] }  
...
```

# Exemple 3

Donner le nombre de croisières par skipper



# Exemple 4

On suppose la création d'une collection croisières

Donner le total des salaires des skippers par ville de destination des croisières

```
db.skippers.aggregate([
  {
    $lookup: {
      from: "croisières",
      localField: "SKNUM",
      foreignField: "SKNUM",
      as: "croisiere_details"
    },
    { $unwind: "$croisiere_details" }
  },
  {
    $group: {
      _id: "$croisiere_details.ARRPORT",
      TotalSalaires: { $sum: "$salaire" }
    }
  },
  {
    $project: {
      _id: 0,
      Destination: "$_id",          // Affiche la destination
      TotalSalaires: 1               // Affiche le total des salaires
    }
  }
]);
```

création d'un document skippers pour chaque élément du tableau croisiere\_details

# Exemple 4

Donner le total des salaires des skippers par ville de destination des croisières

```
{ "SKNUM": "1",
  "SKNOM": "JEAN",
  "SKPORT": "AJACCIO",
  "salaire": 3000 }

{ "SKNUM": "2",
  "SKNOM": "PAUL",
  "SKPORT": "BASTIA",
  "salaire": 2500 }

{ "SKNUM": "3",
  "SKNOM": "LAURA",
  "SKPORT": "ANTIBES",
  "salaire": "aucun" }

{ "SKNUM": "4",
  "SKNOM": "PIERRE",
  "SKPORT": "AJACCIO",
  "salaire": 1500 }
```

\$lookup  
→

```
{ "SKNUM": "1",
  "SKNOM": "JEAN", "SKPORT": "AJACCIO",
  "salaire": 3000, "croisiere_details": [
    {
      "CROISNUM": "C001", "SKNUM": "1", "BATNUM": "B002",
      "DEPPORT": "BASTIA", "ARRPORT": "CALVI",
      "DEPDATE": "2024-07-10", "ARRDATE": "2024-07-11"
    },
    {
      "CROISNUM": "C003", "SKNUM": "1", "BATNUM": "B003",
      "DEPPORT": "AJACCIO", "ARRPORT": "BASTIA",
      "DEPDATE": "2024-08-21", "ARRDATE": "2024-08-22"
    },
    {
      "CROISNUM": "C005", "SKNUM": "1", "BATNUM": "B002",
      "ARRPORT": "CALVI", "DEPDATE": "2024-10-10",
      "ARRDATE": "2024-10-11" } ] }
...
```

# Exemple 4

Donner le total des salaires des skippers par ville de destination des croisières

```
{ "SKNUM": "1",
  "SKNOM": "JEAN", "SKPORT": "AJACCIO",
  "salaire": 3000,
  "croisiere_details": [
    { "CROISNUM": "C001", "SKNUM": "1", "BATNUM": "B002", "DEPPORT": "BASTIA", "ARRPORT": "CALVI",
      "DEPDATE": "2024-07-10", "ARRDATE": "2024-07-11" },
    { "CROISNUM": "C003", "SKNUM": "1", "BATNUM": "B003", "DEPPORT": "AJACCIO", "ARRPORT": "BASTIA",
      "DEPDATE": "2024-08-21", "ARRDATE": "2024-08-22" },
    { "CROISNUM": "C005", "SKNUM": "1", "BATNUM": "B002", "ARRPORT": "CALVI", "DEPDATE": "2024-10-10",
      "ARRDATE": "2024-10-11" } ] }
```

\$unwind



```
{ "SKNUM": "1",
  "SKNOM": "JEAN", "SKPORT": "AJACCIO",
  "salaire": 3000,
  "croisiere_details": [
    { "CROISNUM": "C001", "SKNUM": "1", "BATNUM": "B002", "DEPPORT": "BASTIA", "ARRPORT": "CALVI",
      "DEPDATE": "2024-07-10", "ARRDATE": "2024-07-11" }
```

```
{ "SKNUM": "1",
  "SKNOM": "JEAN", "SKPORT": "AJACCIO",
  "salaire": 3000,
  "croisiere_details": [
    { "CROISNUM": "C003", "SKNUM": "1", "BATNUM": "B003", "DEPPORT": "AJACCIO", "ARRPORT": "BASTIA",
      "DEPDATE": "2024-08-21", "ARRDATE": "2024-08-22" },
    { "CROISNUM": "C005", "SKNUM": "1", "BATNUM": "B002", "ARRPORT": "CALVI", "DEPDATE": "2024-10-10",
      "ARRDATE": "2024-10-11" } ] }
```

...

# Exemple 4

Donner le total des salaires des skippers par ville de destination des croisières

```
{ "SKNUM": "1",  
  "SKNOM": "JEAN", "SKPORT":  
  "AJACCIO",  
  "salaire": 3000,  
  "croisiere_details":  
    { "CROISNUM": "C001", "SKNUM": "1",  
      "BATNUM": "B002", "DEPPORT":  
      "BASTIA", "ARRPORT": "CALVI",  
      "DEPDATE": "2024-07-10", "ARRDATE":  
      "2024-07-11" } }
```

```
{ "SKNUM": "1",  
  "SKNOM": "JEAN", "SKPORT":  
  "AJACCIO",  
  "salaire": 3000,  
  "croisiere_details":  
    { "CROISNUM": "C003", "SKNUM": "1",  
      "BATNUM": "B003", "DEPPORT":  
      "AJACCIO", "ARRPORT": "BASTIA",  
      "DEPDATE": "2024-08-21", "ARRDATE":  
      "2024-08-22" },  
  ... }
```

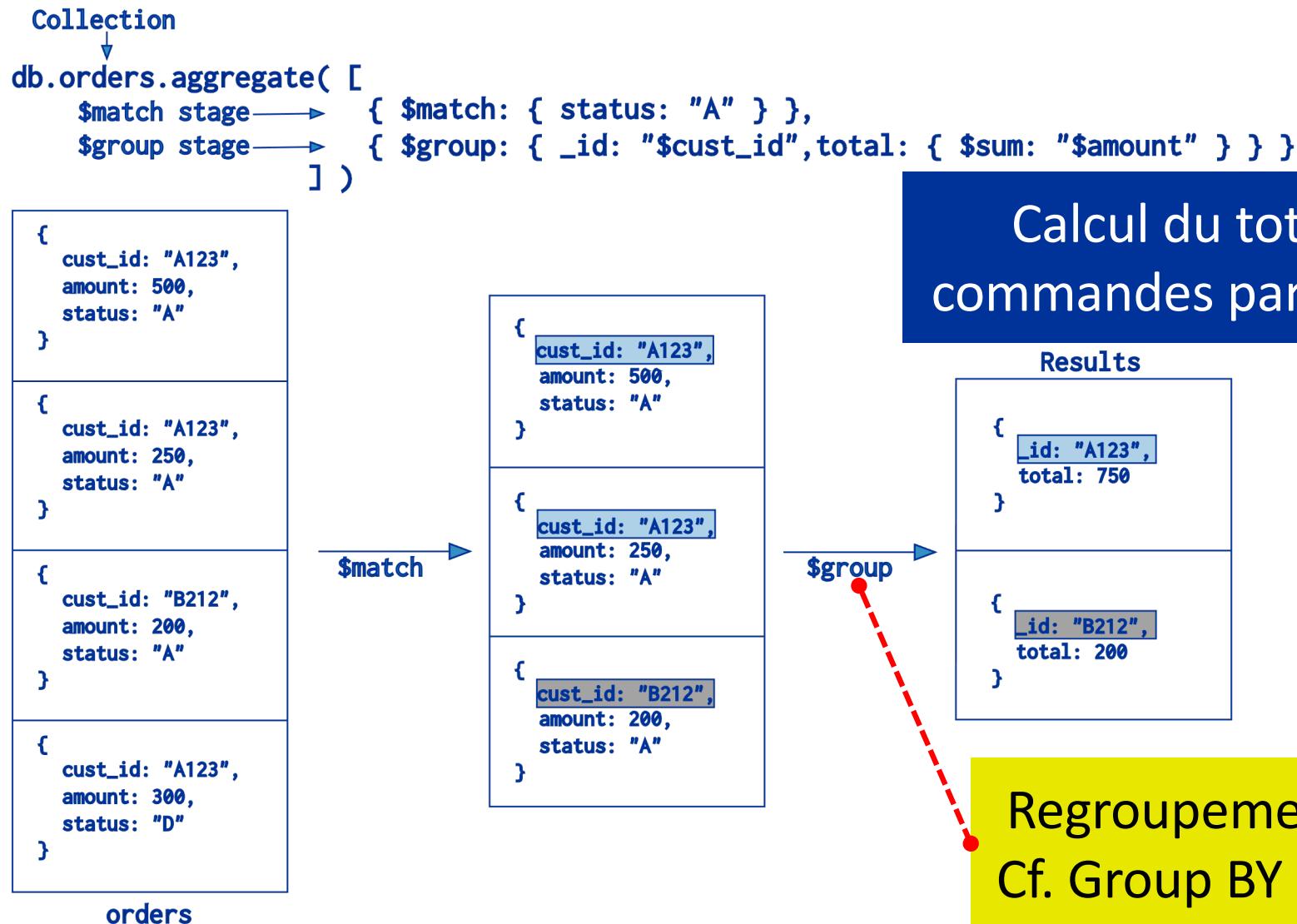
\$group  
→

```
{ "_id": "CALVI",  
  "TotalSalaires": 4500 }  
  
{ "_id": "BASTIA",  
  "TotalSalaires": 2500 }  
  
{ "_id": "AJACCIO",  
  "TotalSalaires": 2500 }  
  
{ "_id": "MARSEILLE",  
  "TotalSalaires": 7500 }
```

\$project

```
{ "Destination": "CALVI",  
  "TotalSalaires": 4500 }  
  
{ "Destination": "BASTIA",  
  "TotalSalaires": 2500 }  
  
{ "Destination": "AJACCIO",  
  "TotalSalaires": 2500 }  
  
{ "Destination": "MARSEILLE",  
  "TotalSalaires": 7500 }
```

# Autres exemples



# A vous de jouer (5)..

- Définissez les commandes permettant de réaliser les actions suivantes:
  1. Calculer le salaire moyen des skippers
  2. Compter le nombre de skippers par port
  3. Trouver le skipper avec le salaire le plus élevé et le plus bas dans chaque port
  4. Donner pour chaque skipper, son nom et le nombre de croisières au départ de Bastia qu'il effectue
  5. Donner la liste des ports d'arrivée pour chaque nom de skipper
    - (*utiliser \$push ou \$addSet dans le \$group*)
  6. Calculer le total des jours en mer pour chaque skipper
    - (*utiliser \$sum et \$dateDiff dans le \$group*)



# A vous de jouer (5).. Correction

2. Compter le nombre de skippers par port



```
db.skippers.aggregate([
  {
    $group: {
      _id: "$skport",
      NumberOfSkippers: { $sum: 1 }
    }
  }
]);
```

# Une alternative plus ancienne (obsolète!) : Technique MapReduce

- Méthode pour effectuer des calculs distribués sur de grandes quantités de données.
- Fonction **map** :
  - Traite chaque document de la collection,
  - retourne **une clé et une valeur** pour chaque élément souhaité.
- Fonction **reduce**:
  - Prend les clés retournées par la fonction map et
  - « Réduit » les données de toutes les valeurs ayant la même clé.
- Collection de sortie : nouvelle collection

valeurs intermédiaires

« réduit» signifie faire un calcul qui renvoie une seule valeur

# MapReduce : Avantages/Inconvénients

- **Avantages :**

Utile pour traiter de grandes quantités de données réparties sur plusieurs machines.

- **Inconvénients :**

Moins efficace que l'agrégation pour la plupart des opérations simples en raison de son coût de démarrage plus élevé et de son exécution généralement plus lente.

# Exemple1 avec MapReduce

- Calculer le salaire total de tous les skippers sauf ceux qui n'ont pas de salaire défini

```
function map() {  
    // On vérifie que le champ 'salaire' existe et n'est  
    // pas 'aucun'.  
    if (this.salaire && this.salaire !== "aucun") {  
        emit("totalSalaire", salaire);  
    }  
}  
  
function reduce(cle, valeurs) {  
    return Array.sum(valeurs);  
    // Additionne toutes les valeurs pour obtenir le  
    // salaire total.  
}
```

# Exemple1 avec MapReduce (2)

```
var res=db.skippers.mapReduce(  
  map, reduce,  
  {  
    out: { inline: 1 },  
    // Les résultats seront retournés directement  
    // dans la sortie de la commande.  
    query: { salaire: { $exists: true, $ne: "aucun" } }  
    // Filtrage initial optionnel pour optimiser.  
  }  
);  
printjson(res); // Affichage du résultat
```

Ici on pourrait choisir une collection de sortie (ex: « ColSalaires »)

# Exemple2 avec MapReduce

- Trouver le salaire total par port de résidence

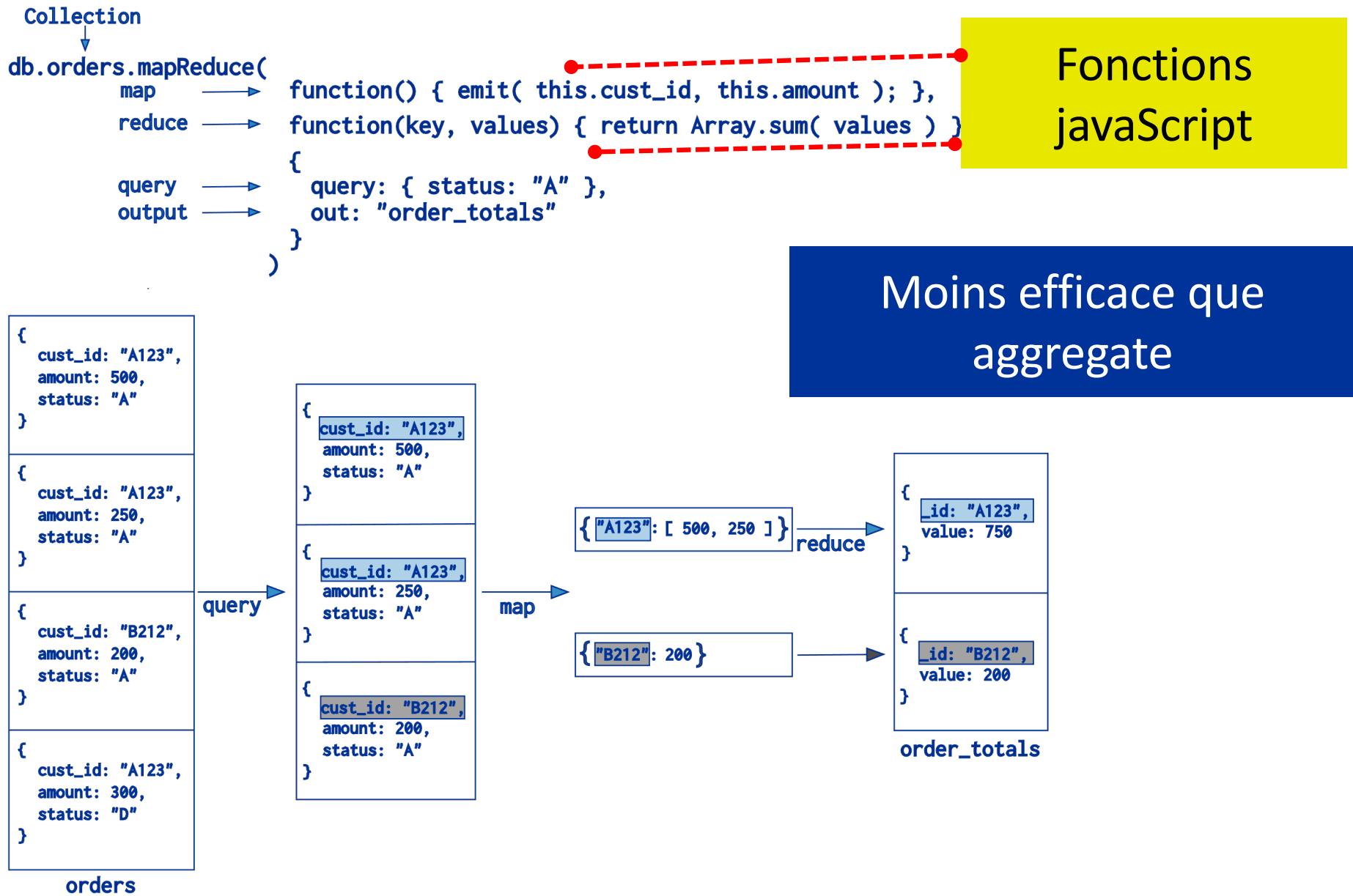
```
function map2() {  
    if (this.salaire && this.salaire !== "aucun") {  
        emit(this.skport, salaire);  
    }  
}  
//Fonction reduce de l'exemple 1
```

La clé est ici le port

# Exemple2 avec MapReduce (2)

```
res= db.skippers.mapReduce(  
    map2, reduce,  
    {  
        out: { inline: 1 },  
        // Les résultats seront retournés directement  
        // dans la sortie de la commande.  
        query: { salaire: { $exists: true, $ne: "aucun" } ,  
                 skport: { $exists: true }}  
        // Filtrage initial optionnel pour optimiser.  
    }  
);  
printjson(res); // Affichage du résultat
```

# Autre exemple : Map/Reduce



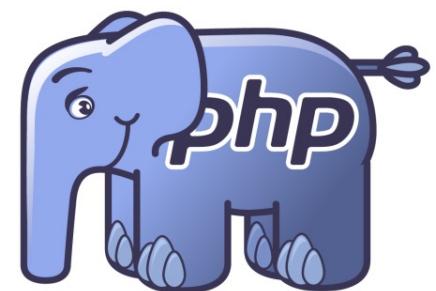
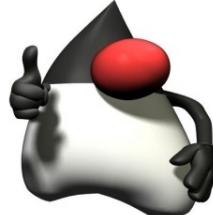
# A vous de jouer (6)..



- Définissez à l'aide de la technique map-reduce les commandes permettant de réaliser les actions suivantes:
  1. Calculer le salaire moyen des skippers
  2. Compter le nombre de skippers par port
  3. Trouver le skipper avec le salaire le plus élevé et le plus bas dans chaque port



# Manipuler MongoDB dans un langage Hôte



# Installations ...

- Télécharger un driver MongoDB pour le langage choisi (ex: php, Java)
  - Driver Java:
    - <http://mvnrepository.com/artifact/org.mongodb/mongo-java-driver>

Driver = bibliothèque (library)  
ensemble de classes

- Lier le driver à votre projet

# Principes de manipulation

- Création et utilisation de plusieurs Objets définis sur les classes offertes par le driver
- Les étapes sont similaires à la manipulation à d'une BD relationnelle
  - connexion, manipulation, fermeture
  - variables de type collection avec parcours itératif
- Les méthodes find possèdent parfois des paramètres spécifiques

# Principales étapes

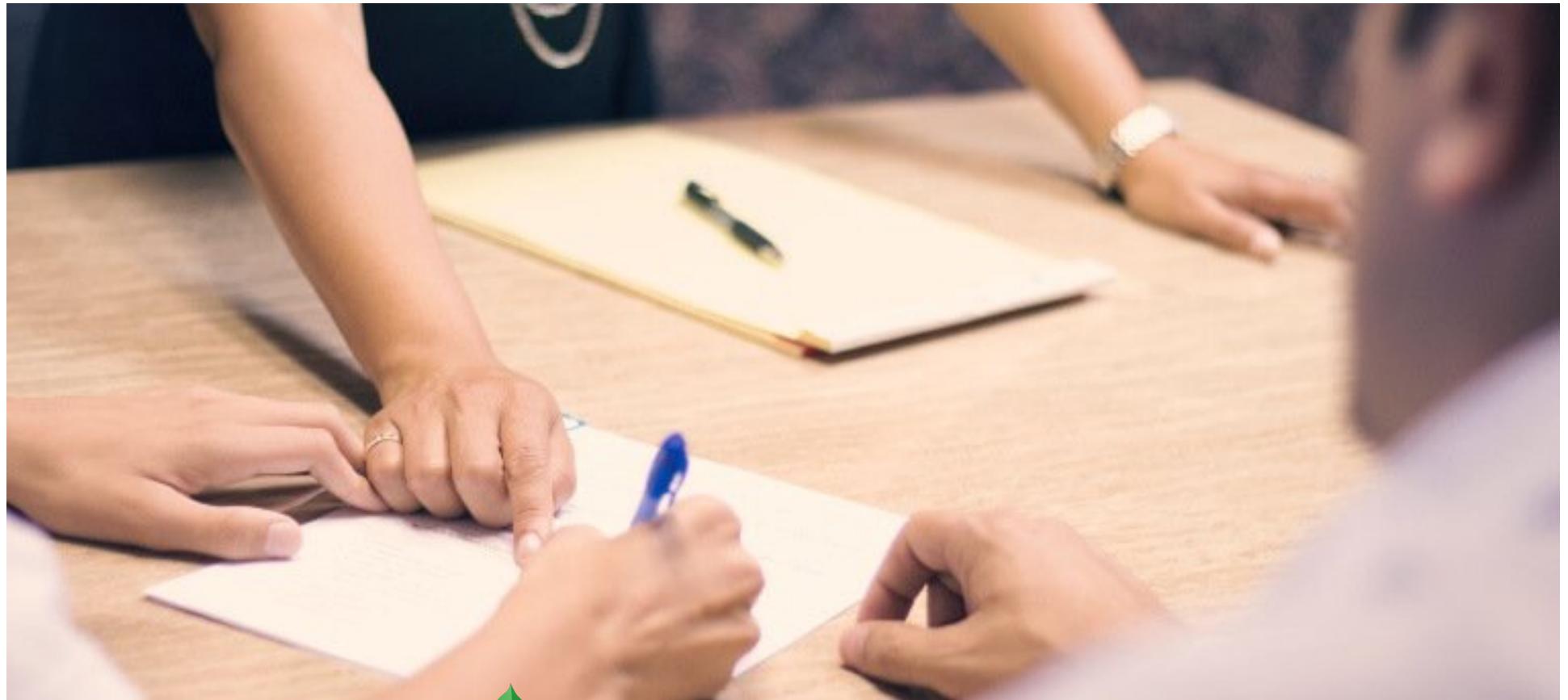
1. Créer un objet de connexion au serveur MongoDB (**MongoClient** en Java par ex)
2. Créer un objet BD associé à une BD spécifique
3. Manipuler la BD.....
  - insérer des documents (identifier les types apparaissant en paramètre des méthodes d'insertion)
  - mettre à jour ....
  - récupérer des documents (méthodes de type find)
4. Fermer les objets

# Quelques liens



- MongoDB et php
  - Documentation des classes
    - <https://www.php.net/manual/fr/set.mongodb.php>
  - Tutoriel
    - [https://www.tutorialspoint.com/mongodb/mongodb\\_php.htm](https://www.tutorialspoint.com/mongodb/mongodb_php.htm)
- MongoDB et java
  - Tutoriels
    - <http://mongodb.github.io/mongo-java-driver/3.0/driver/getting-started/quick-tour>
    - <http://www.torrefacteurjava.fr/content/cours-mongodb-semaine-2-crud>

# Comment concevoir une BD MongoDB?



mongoDB

# Comment concevoir une BD mongoDB?

- Comment organiser les collections?
  - Aucune méthode systématique
  - Conception au cas par cas en fonction des utilisations prévues de la BD
    - critères de recherche prévus
    - mises à jour envisagées
- Faut-il se conformer au schéma relationnel?
  - En général non
  - Mais il ne faut pas systématiquement s'y opposer
    - Problèmes de Mises à jour!!

# Un exemple de BD relationnelle

ETUDIANT

etu_id	nom	prenom
1	Paoli	pascal
2	Bonaparte	Napoleon

COURS

cours_id	titre	nombreHeures
1	Politique	50
2	Philosophie	75

INSCRIPTION

cours_id	etu_id
1	1
1	2
2	1

# Traduction en MongoDB

## Solution 1: Une collection cours

document cours1

```
{titre: "Politique",  
nombreHeures : 50,  
etudiants :  
  [{nom: "Paoli", prenom: "Pascal"} ,  
   {nom: "Bonaparte", prenom:"Napoleon"} ] } }
```

valeur de type  
tableau

document cours2

```
{titre: "Philosophie",  
nombreHeures : 75,  
etudiants :  
  [{nom: "Paoli", prenom: "Pascal"} ] }
```

# Traduction en MongoDB

## Solution 2: Une collection etudiants

```
{nom: "Paoli",
prenom: "Pascal" ,          document etudiant1
cours: [
    { titre:"Politique" , nombreHeures : 50 } ,
    { titre:"Philosophie" , nombreHeures : 75} ]
}
```

```
{nom: "Bonaparte",
prenom: "Napoleon" ,          document etudiant2
cours: [
    { titre:"Politique" , nombreHeures : 50 } ]
```

# Traduction en MongoDB

Comment choisir entre les deux solutions?

- Principaux critères de recherche
  - titre de cours ?
  - nom d'auteur?
- Champs mis à jour le plus fréquemment
  - champs caractéristiques des cours (ex: nb heures)?
  - champs caractéristiques des auteurs?

On peut aussi choisir de définir plusieurs collections dans le style relationnel

# A vous de jouer (6)..

SKIPPER
SKNUM
SKNOM
SKPORT
SALAIRE
1 JEAN AJACCIO 3000
2 PAUL AJACCIO 2000
3 PIERRE ANTIBES 1500
4 MARIE BASTIA 1500

BATEAU	BATNUM	BATNOM	BATPORT	CAPACITE
B001	LIBERTE	ANTIBES	10	
B002	LOUISIANE	BASTIA	10	
B003	KALISTE	BASTIA	6	
B004	INDEPENDANCE	CALVI	6	

CROISIERE
CROISNUM
SKNUM
BATNUM
DEPPORT
ARRPORT
C001 1 B002 BASTIA CALVI
C002 2 B002 ANTIBES AJACCIO
C003 1 B003 AJACCIO BASTIA
C004 3 B004 CALVI MARSEILLE

- Supprimez votre collection skippers et redéfinissez la afin d'y inclure les informations contenues dans les deux autres tables Bateaux et Croisieres.

solution skippers.js



attention : Vous ne devez définir qu'une seule collection avec un document par skipper

# A vous de jouer (6)..

## BD croisières en MongoDB

SKIPPERS				
SKNUM	SKNOM	SKPORT	SALAIRE	
1	JEAN	AJACCIO	3000	
2	PAUL	AJACCIO	2000	
4	MARIE	BASTIA	1500	
3	PIERRE	ANTIBES	1500	
BATEAUX				
BATNUM	BATNOM	BATPORT	CAPACITE	
B001	LIBERTE	ANTIBES	10	
B002	LOUISIANE	BASTIA	10	
B003	KALISTE	BASTIA	6	
B004	INDEPENDANCE	CALVI	6	
CROISIERES				
CROISNUM	SKNUM	BATNUM	DEPORT	ARRPORT
C001	1	B002	BASTIA	CALVI
C002	2	B002	ANTIBES	AJACCIO
C003	1	B003	AJACCIO	BASTIA
C004	3	B004	ANTIBES	AJACCIO

### collectionSKIPPERS

```

var skipper1={
  sknum:'1',
  sknom:'JEAN',
  skport:'AJACCIO',
  salaire:3000,
  croisières: [
    {croisnum : 'C001',
     deport : 'BASTIA',
     arrport : 'CALVI',
     bateau : {
       batnum:'B002',
       batnom:'LOUISIANE',
       batport:'BASTIA',
       capacite:10
     },
    {croisnum : 'C003',
     deport : 'AJACCIO',
     arrport : 'BASTIA',
     bateau : {
       batnum:'B003',
       batnom:'KALISTE',
       batport:'BASTIA',
       capacite:6
     }
   ]}]
};

var skipper2={
  sknum:'2',
  sknom:'PAUL',
  skport:'AJACCIO',
  salaire:2000,
  croisières: [
    {croisnum : 'C002',
     deport : 'ANTIBES',
     arrport : 'AJACCIO',
     bateau : {
       batnum:'B002',
       batnom:'LOUISIANE',
       batport:'BASTIA',
       capacite:10
     }
   ]];
};

var skipper3={
  sknum:'3',
  sknom:'PIERRE',
  skport:'ANTIBES',
  salaire: 1500,
  croisières: [
    {croisnum : 'C004',
     deport : 'ANTIBES',
     arrport : 'AJACCIO',
     bateau : {
       batnum:'B004',
       batnom:'INDEPENDANCE',
       batport:'CALVI',
       capacite:6
     }
   ]];
};

```

Est-ce correct ?

# A vous de jouer (7)..

- Définissez les interrogations suivantes:

1. Afficher les noms des skippers qui n'effectuent aucune croisière

\*\*\*\* Liste des skippers sans croisières \*\*\*\*  
MARIE

2. Afficher les noms des skippers qui conduisent le bateau numéro B002

\*\*\*\* Liste des skippers qui conduisent le bateau numéro B002 \*\*\*\*  
JEAN  
PAUL

3. Afficher les numéros des croisières au départ d'Ajaccio

\*\*\*\* Liste numeros des croisières au départ d'Ajaccio \*\*\*\*  
C003

4. Afficher les noms des bateaux barrés par le skipper numéro 1

\*\*\*\* Liste des noms des bateaux barrés par le skipper numéro 1 \*\*\*\*  
LOUISIANE  
KALISTE

5. Afficher tous les numéros de bateaux

\*\*\*\* Liste numeros de bateaux \*\*\*\*  
B002  
B003  
B004



# A vous de jouer (8)..



- Que pensez-vous des résultats obtenus dans la requête 5 par rapport aux tables initiales (cf. page 78)?
- Quel serait le résultat d'une requête recherchant les numéros des bateaux n'effectuant aucune croisière?
- A quelle anomalie évoquée au CH1-normalisation sommes-nous confrontés ici?
- Si la collection avait été construite à partir des bateaux, est ce que le problème aurait été résolu?

# A vous de jouer (9)..



- Définissez à présent une BD Mongo structurée comme la BD initiale Croisiere :
  - composée de 3 collections Skippers, Bateaux et Croisieres
- Définissez les requetes précédentes (1 à 5) dans cette nouvelle base
- Quelles remarques pouvez-vous faire?

solution  
`skippersBateauxCroisieres.js`

# Notion de schéma Mongo DB

- Depuis la version 3.6, MongoDB propose des outils de validation de schéma JSON.
- <https://docs.mongodb.com/manual/core/schema-validation/>
- Utilisation de la bibliothèque mongoose sous nodeJs.

# Bilan sur une solution MongoDB

## ■ Avantages

- performances accrues pour les interrogations portant sur les critères « bien placés »
  - permet d'éviter les jointures
- absence de valeurs nulles
- meilleure connexion avec la programmation

## ■ Inconvénients

- conception: attention danger!! aucun garde-fou!
- les tâches de programmation peuvent être plus complexes

# Liens : MongoDB



- Le cours de Philippe Rigaux (professeur au CNAM) sur le NoSQL en général et une bonne introduction à MongoDB  
<http://b3d.bdpedia.fr/intro.html>
- Comparaison Oracle/MongoDB (par Mongodb...)  
<https://www.mongodb.com/compare/mongodb-oracle?lang=fr-fr>