



# Bases de Données Avancées

**TD/TP : NoSQL- MongoDB**

- USTHB Master 01 IL-

M. AZZOUZ

Dernière mis à jour :Mai 2020

## Exercice 01

a. Créer une nouvelle base de données nommée info et vérifiez qu'elle est sélectionnée.

❑ Syntaxe de création de base de données MongoDB est la suivante: **use DATABASE\_NAME**.

❑ La clause **use**: Si la base de données n'existe pas le SGBD va créer une base de données. Sinon le SGBD va se connecter à la base de données spécifiée. D'où use permet de créer et se connecter à une BD.

```
>use info
```

```
switched to db info
```

```
> db
```

```
info
```

❑ Si vous voulez voir toutes les bases de données, vous pouvez utiliser la commande **show dbs**.

## Exercice 01

❑ La syntaxe de suppression d'une base de données MongoDB est la suivante: **db.dropDatabase()**.

❑ L'instruction permet de supprimer de la base de données actuellement connectée. Le SGBD connecte par défaut la base de données test.

❑ **Exemple:**

```
> use mydb
```

```
switched to db mydb
```

```
> db.dropDatabase()
```

```
{ "ok" : 1 }
```

## Exercice 01

Dans MongoDB, comme nous le verrons par la suite, nous utilisons un formalisme de type **db.collection.fonction()** :

- **db** représente la base de données choisie grâce à la commande use (ce mot clé est non modifiable).
- **collection** représente la collection dans laquelle nous allons effectuer l'opération, et doit donc correspondre à une des collections présentes dans la base.
- **fonction()** détermine l'opération à effectuer sur la collection.

## Exercice 01

**b. Créer une nouvelle collection nommée produits et y insérer le document suivant:**

➤ La syntaxe de création de la collection est **db.createCollection("CollectionName")**

```
> db.createCollection("produit")
```

```
{ "ok" : 1 }
```

➤ Pour afficher les collections d'une BD, on utilise : **show collections**

➤ Plusieurs fonctionnalités sont disponibles pour insérer des documents: **Insert**, **InsertOne** et **InsertMany**.

➤ Il est possible d'insérer directement le document ou le définir puis l'insérer.

```
db.produit.insert({"nom": "Macbook Pro", "fabriquant": "Apple", "prix": 11435.99, "options": ["Intel Core i5", "Retina Display", "Long life battery"]})
```

## Exercice 01

**c. et d. Ajout d'autres documents:**

```
➤ document1={ "nom" : "Macbook Air",  
  "fabriquant" : "Apple",  
  "prix" : 125794.73, "ultrabook" : true,  
  "options" : ["Intel Core i7", "SSD", "Long life  
battery"] }
```

```
db.produit.insert(document1)
```

```
➤ document2={ "nom" : "Thinkpad X230",  
  "fabriquant" : "Lenovo",  
  "prix" : 114358.74, "ultrabook": true,  
  options: ["Intel Core i5", "SSD", "Long life  
battery"] }
```

```
db.produit.insert( document2)
```

## Exercice 01

### e. Requêtes:

❑ Récupérer tous les produits.

➤ Pour visualiser tous les documents de la collection on utilise la fonction **find()**.

**db.produit.find()**

➤ L'affichage rendu par **find()** est compact et peu lisible directement. On peut ajouter la fonction **pretty()** pour avoir une présentation propre.

**db.prouduit.find().pretty()**

## Exercice 01

### e. Requêtes:

La fonction `find()` sans paramètre, elle renvoie l'ensemble des documents. Mais celle-ci peut aussi prendre deux paramètres :

- Les critères de sélection des documents (condition)
- Les choix d'items des documents à afficher (projection)

Ces deux paramètres doivent être écrits sous la forme d'objets JSON.



## Exercice 01

### e. Requêtes:

❑ Récupérer le premier produit.

➤ `db.produit.findOne()`

➤ ou bien `db.produit.find()[0]`

❑ Une autre fonction très utile pour mieux appréhender les données est de lister les valeurs prises par les différents items de la collection sans les doublons, grâce à **`distinct()`**.

➤ `db.produit.distinct("fabriquant")`

```
[ "Apple",  
  "Lenovo"]
```

## Exercice 01

### e. Requêtes:

□ Trouver l'id du **Thinkpad X230** et faire la requête pour récupérer ce produit avec son id.

➤ `db.produit.find(`  
`{ "nom": "Thinkpad X230" },` ← Critères de sélection  
`{"_id":1})` ← Projection: les items à garder

Si l'on désire n'afficher que certains éléments, il est possible d'ajouter un deuxième argument spécifiant les items que l'on veut (avec 1) ou qu'on ne veut pas (avec 0).

- **Résultat:** `{"_id": ObjectId("5ea74afa390a5ba392b3c7fd" )}`

- `db.produit.find(`  
`{ "_id": ObjectId("5ea74afa390a5ba392b3c7fd") })`

## Exercice 01

### e. Requêtes:

❑ Récupérer les produits dont le prix est supérieur à 13723 DA.

➤ Pour les comparaisons, nous disposons des opérateurs **\$eq** (equal), **\$gt** (greater than), **\$gte** (greater than or equal), **\$lt** (less than), **\$lte** (less than or equal) et **\$ne** (not equal).

```
-db.produit.find(  
  {"prix": {$gt: 13723}},  
  {"_id":0})
```

➤ En plus de ces comparaisons simples, nous disposons d'opérateurs de comparaisons à une liste : **\$in** (présent dans la liste) et **\$nin** (non présent dans la liste).

## Exercice 01

### e. Requêtes:

❑ Récupérer le premier produit ayant le champ ultrabook à true.

Pour limiter le nombre de documents renvoyés par la fonction `find()` en lui ajoutant la fonction **`limit()`**, comme ici où nous nous restreignons au premier résultat.

```
-db.produit.find(  
  {"ultrabook" : true},  
  
  {"nom":1, "_id":0}  
  
).limit(1)
```

## Exercice 01

### e. Requêtes:

□ Une autre opération classique est le tri des résultats, réalisable avec la fonction **sort()**. On doit indiquer les items de tri et leur attribuer une valeur de 1 pour un tri ascendant et une valeur de -1 pour un tri descendant. On affiche ici les produits dans l'ordre croissant de leur prix.

```
db.produit.find().sort({ "prix" : 1 })
```

Idem que précédemment, mais dans l'ordre décroissant

```
db.produit.find().sort({ "prix" : -1 })
```

## Exercice 01

### e. Requêtes:

- ❑ Utilisation des expressions régulières
- ❑ Récupérer le premier produit dont le nom contient Macbook.

```
db.produit.find({"nom": /Macbook/}).limit(1)
```

- ❑ Récupérer les produits dont le nom commence par Macbook

```
db.produit.find({"nom": /^Macbook/},  
{"_id":0, "nom":1, "prix":1})  
.sort({"nom" : 1 })
```

- ❑ Récupérer les produits dont le nom se termine par Macbook: `db.produit.find({"nom": /Macbook$/}).limit(1)`

## Exercice 01

### ❑ Mise à jour d'un document dans une collection

Dans cet exemple on a modifié le prix de produit de nom Macbook Air.

```
db.produit.updateOne({"nom" : "Macbook Air"},  
{$set:{"prix" : 125795}}).
```

### ❑ Mise à jour de plusieurs documents dans une collection

Dans cet exemple on a modifié le fabricant des produits dont le nom commence par Macbook.

```
db.produit.updateMany({"nom":/^Macbook/},  
{$set:{"fabriquant" : "Apple Entreprise"}})
```

## Exercice 01

❑ Opération de mise à jour avec remplacement d'un document dans une collection:

```
db.produit.replaceOne(  
  {"nom" : "Macbook Pro"},  
  {"nom" : "Macbook Pro",  
    "fabriquant" : "Apple",  
    "prix" : 11436,  
    "options" : [  
      "Intel Core i5",  
      "Retina Display",  
      "Long life battery"  
    ]})
```

- Le document remplacé conserve le même identifiant de l'objet **id**.



## Exercice 01

### e. Requêtes:

❑ Supprimer les deux produits dont le fabricant est Apple

```
-db.produit.remove({"fabriquant": "Apple"})
```

❑ Supprimer le Thinkpad X230 en utilisant uniquement son id

```
➤ db.produit.find({"nom": "Thinkpad X230"}, {"_id": 1, "nom": 1})
```

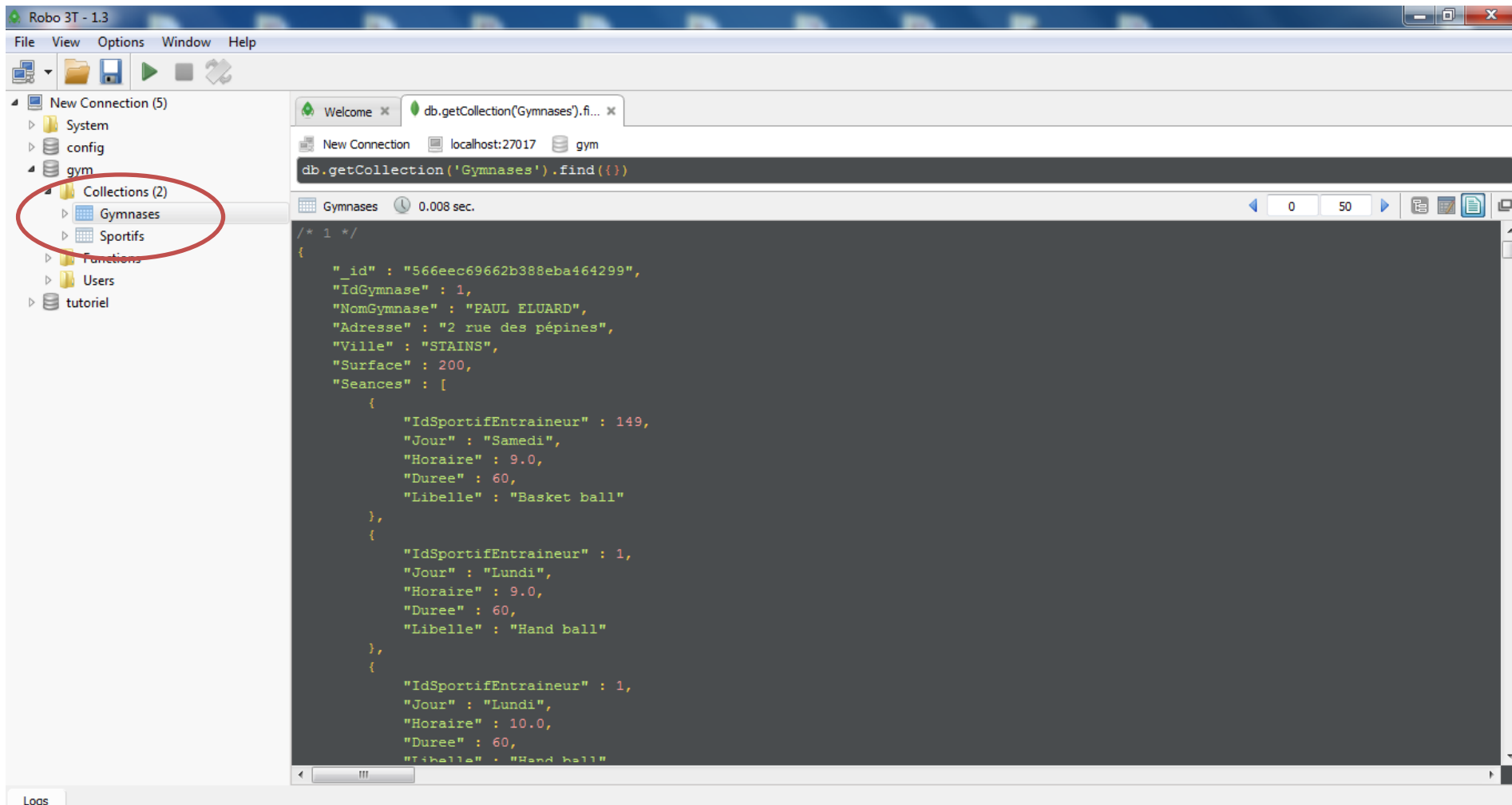
➤ **Résultat:**

```
{ "_id" : ObjectId("5ea74afa390a5ba392b3c7fd"), "nom" : "Thinkpad X230" }
```

```
➤ db.produit.remove({"_id": ObjectId("5ea74afa390a5ba392b3c7fd")})
```

# Exercice 02

## Vue d'ensemble sur la BD gym:



The screenshot shows the Robo 3T - 1.3 interface. On the left, the 'Collections (2)' folder is expanded, and the 'Gymnases' collection is selected. The main window displays the following MongoDB query and its results:

```
db.getCollection('Gymnases').find({})
```

Execution time: 0.008 sec.

```
/* 1 */
{
  "_id" : "566eec69662b388eba464299",
  "IdGymnase" : 1,
  "NomGymnase" : "PAUL ELUARD",
  "Adresse" : "2 rue des pépines",
  "Ville" : "STAINS",
  "Surface" : 200,
  "Seances" : [
    {
      "IdSportifEntraîneur" : 149,
      "Jour" : "Samedi",
      "Horaire" : 9.0,
      "Duree" : 60,
      "Libelle" : "Basket ball"
    },
    {
      "IdSportifEntraîneur" : 1,
      "Jour" : "Lundi",
      "Horaire" : 9.0,
      "Duree" : 60,
      "Libelle" : "Hand ball"
    },
    {
      "IdSportifEntraîneur" : 1,
      "Jour" : "Lundi",
      "Horaire" : 10.0,
      "Duree" : 60,
      "Libelle" : "Hand ball"
    }
  ]
}
```

# Exercice 02

## Collection gymnases:

The screenshot shows the Robo 3T 1.3 interface. The left sidebar displays a project structure with folders for System, config, gym, Collections (2), Sportifs, Functions (0), Users, and tutorial. The main window shows a query: `db.getCollection('Gymnases').find({})` executed against the 'gym' database on localhost:27017. The results are displayed in a table with columns for Key, Value, and Type.

Key	Value	Type
▲ (1) 566eec69662b388eba464299	{ 7 fields }	Object
" _id	566eec69662b388eba464299	String
# IdGymnase	1	Int32
" NomGymnase	PAUL ELUARD	String
" Adresse	2 rue des pépines	String
" Ville	STAINS	String
# Surface	200	Int32
▲ Seances	[ 20 elements ]	Array
▲ [0]	{ 5 fields }	Object
# IdSportifEntraîneur	149	Int32
" Jour	Samedi	String
## Horaire	9,0	Double
# Duree	60	Int32
" Libelle	Basket ball	String
▲ [1]	{ 5 fields }	Object
# IdSportifEntraîneur	1	Int32
" Jour	Lundi	String
## Horaire	9,0	Double
# Duree	60	Int32
" Libelle	Hand ball	String
▶ [2]	{ 5 fields }	Object
▶ [3]	{ 5 fields }	Object
▶ [4]	{ 5 fields }	Object
▶ [5]	{ 5 fields }	Object

# Exercice 02

## Collection Sportifs:

The screenshot shows the Robo 3T - 1.3 interface. The left sidebar displays a tree view of the database structure, including 'System', 'config', 'gym', 'Collections (2)', 'Gymnases', 'Sportifs', 'Functions', 'Users', and 'tutoriel'. The main window shows a query result for the 'Sportifs' collection. The query is `db.getCollection('Sportifs').find({})`. The result is a document with the following structure:

Key	Value	Type
(2) 566eec5f662b388eba464204	{ 8 fields }	Object
_id	566eec5f662b388eba464204	String
IdSportif	2	Int32
Nom	KERVADEC	String
Prenom	Yann	String
Sexe	M	String
Age	28	Int32
IdSportifConseiller	1	Int32
Sports	{ 3 fields }	Object
Jouer	[ 4 elements ]	Array
[0]	Basket ball	String
[1]	Volley ball	String
[2]	Ping pong	String
[3]	Football	String
Arbitrer	[ 2 elements ]	Array
[0]	Hockey	String
[1]	Football	String
Entraîner	[ 8 elements ]	Array
[0]	Basket ball	String
[1]	Volley ball	String
[2]	Hand ball	String
[3]	Tennis	String
[4]	Hockey	String
[5]	Badmington	String

## Exercice 02

a. Quels sont les sportifs (identifiant, nom et prénom) qui ont un âge entre 20 et 30 ans ?

```
db.Sportifs.find(  
{
```

```
  "Age": { "$gte": 20, "$lte": 30 }  
},
```

La restriction

```
{  
  "_id": 0,  
  "IdSportif": 1,  
  "Nom": 1,  
  "Prenom": 1  
}
```

La Projection

```
)
```

## Exercice 02

**b. Quels sont les gymnases de ville "Villetaneuse" ou de "Sarcelles" qui ont une surface de plus de 400 m<sup>2</sup> ?**

```
db.Gymnases.find(  
  {"Ville":{"$in":["VILLETANEUSE", "SARCELLES"]},  
   "Surface": { "$gt": 400 }  
},  
{  
  "_id": 0,  
  "NomGymnase": 1,  
  "Ville": 1,  
  "Surface": 1  
})
```

## Exercice 02

c. Quels sont les sportifs (identifiant et nom) qui pratiquent du hand ball ?

```
db.Sportifs.find(  
  {  
    "Sports.Jouer": "Hand ball"  
  },  
  {  
    "_id": 0,  
    "IdSportif": 1,  
    "Nom": 1  
  }  
)
```

Pour spécifier un sous-item d'un item, il est nécessaire d'utiliser le formalisme **item.sousitem**.

## Exercice 02

e. Dans quels gymnases et quels jours y a t-il des séances de hand ball ?

```
db.getCollection('Gymnases').find({})
```

Gymnases 0.008 sec. 0 50

```
{
  "_id" : "566eec69662b388eba464299",
  "IdGymnase" : 1,
  "NomGymnase" : "PAUL ELUARD",
  "Adresse" : "2 rue des pépines",
  "Ville" : "STAINS",
  "Surface" : 200,
  "Seances" : [
    {
      "IdSportifEntraîneur" : 149,
      "Jour" : "Samedi",
      "Horaire" : 9.0,
      "Duree" : 60,
      "Libelle" : "Basket ball"
    },
    {
      "IdSportifEntraîneur" : 1,
      "Jour" : "Lundi",
      "Horaire" : 9.0,
      "Duree" : 60,
      "Libelle" : "Hand ball"
    },
    {
      "IdSportifEntraîneur" : 1,
      "Jour" : "Lundi",
      "Horaire" : 10.0,
      "Duree" : 60,
      "Libelle" : "Hand ball"
    }
  ]
}
```

-Chaque document gymnase peut avoir un item **Seances** qui correspond à un tableau des Seances et chaque séance correspond à un sport(item **Libelle**), pour qu'on puisse faire la requête il faut faire le découpage de tableau Seances (rappelez vous de l'opérateur table en SQL3).

- Pour faire la requête on utilise le Framework d'agrégation



## Exercice 02

- ❑ **Framework d'agrégation:** Depuis la version 2.2, MongoDB propose un Framework d'agrégation.
- ❑ Le framework d'agrégation comporte un ensemble large d'opérateurs :
  - **\$project** : redéfinition des documents (si nécessaire).
  - **\$match** : restriction sur les documents à utiliser.
  - **\$group** : regroupements et calculs à effectuer.
  - **\$sort** : tri sur les agrégats.
  - **\$unwind** : découpage de tableaux.
- ❑ Une requête d'agrégation est de la forme `db.collection.aggregate( [ { ... }, { ...}, ... ] )`.

## Exercice 02

e. Dans quels gymnases et quels jours y a t-il des séances de hand ball ?

```
db.Gymnases.aggregate([
  { $unwind : "$Seances" },
  { $match: { "Seances.Libelle" : "Hand ball" }},
  { $group: {
    "_id": {
      "Nom": "$NomGymnase ",
      "Ville": "$Ville",
      "Jour": { $toLower: "$Seances.Jour" }
    },
    "nb": { $sum: 1 }
  }},
  { $sort: {
    "_id.Ville": 1,
    "_id.Nom": 1,
    "nb": -1
  }}
])
```

**\$unwind** : permet d'éclater le tableau en autant de documents que de séances.


L'opérateur **\$group** doit contenir obligatoirement une clé de groupement (**\_id**), puis une clé (**nb**) à laquelle on associe la fonction d'agrégation ici (**\$sum**)

## Exercice 02

f. Quels sportifs (identifiant et nom) ne pratiquent aucun sport ?

```
db.Sportifs.find(  
  {  
    "Sports" : { "$exists" : false }  
  },  
  {  
    "_id": 0,  
    "Nom": 1  
  }  
)
```

Si l'on cherche à tester la présence ou non d'un item, on utilise l'opérateur **\$exists** (avec true si on teste la présence, et false l'absence).



## Exercice 02

g. Quels gymnases n'ont pas de séances le dimanche ?

```
db.Gymnases.find(  
  {"Seances.Jour": { "$nin": [ "dimanche", "Dimanche" ] }  
  },  
  { "_id": 0,  
    "NomGymnase": 1,  
    "Ville": 1,  
    "Seances.Jour": 1  
  }  
)
```

## Exercice 02

**h. Quels gymnases ne proposent que des séances de basket ball ou de volley ball ?**

il existe des opérateurs logiques : *\$and*, *\$or* et *\$nor*. Ces trois opérations prennent un tableau de critères comme valeur.

```
db.Gymnases.find(  
  {  
    "$nor": [  
      { "Seances.Libelle": { "$ne": "Basket ball" } },  
      { "Seances.Libelle": { "$ne": "Volley ball" } } ]  
  },  
  {  
    "_id": 0,  
    "NomGymnase": 1,  
    "Ville": 1,  
    "Seances.Libelle": 1  
  })
```

**\$nor** effectue une opération logique **NOR** sur un tableau d'une ou plusieurs expressions de requête et sélectionne les documents qui échouent à toutes les expressions de requête du tableau.

## Exercice 02

**i. Quels sont les entraîneurs qui sont aussi joueurs ?**

```
db.Sportifs.find(  
  {  
    "Sports.Jouer" : { "$exists" : true },  
    "Sports.Entraîner" : { "$exists" : true }  
  },  
  {  
    "_id": 0,  
    "Nom": 1  
  }  
)
```

## Exercice 02

j. Quels sont les sportifs qui sont des conseillers ?

```
db.Sportifs.find(  
  {"IdSportif":{"$in":  
    db.Sportifs.distinct("IdSportifConseiller")}  
  },  
  {  
    "_id": 0,  
    "Nom": 1  
  }  
)
```




Pour récupérer la liste  
de tous les conseillers

## Exercice 02

k. Pour le sportif "Kervadec" quel est le nom de son conseiller ?

```
db.Sportifs.find(  
{  
  "IdSportif":db.Sportifs.findOne(  
    {"Nom": "KERVADEC" }).IdSportifConseiller  
  },  
{  
  "_id": 0,  
  "Sports": 0  
})
```

rechercher id de  
conseiller de  
sportif de nom  
**Kervadec**





## Exercice 02

### 1. Quels entraîneurs entraînent du hand ball et du basket ball ?

```
db.Sportifs.find(
  { "Sports.Entraîner": "Hand ball",
    "Sports.Entraîner": "Basket ball" },
  { "_id": 0,
    "Nom": 1,
    "Sports.Entraîner": 1 }
)
```

---

```
db.Sportifs.find(
  {$and: [
    { "Sports.Entraîner": "Hand ball" },
    { "Sports.Entraîner": "Basket ball" }
  ]
},
  { "_id": 0,
    "Nom": 1,
    "Sports.Entraîner": 1
  }
)
```

## Exercice 02

**m. Quels sont les couples de sportifs (identifiant et nom et prénom de chaque) de même âge ?**

```
Var sportifs1=db.Sportifs.find({},
{"_id":0, "IdSportif":1,"Nom":1,"Prenom":1,"Age":1 }
).toArray();
var sportifs2 =sportifs1;
for(var i = 0; i < sportifs1.length; i++){
for(var j = 0; j < sportifs2.length; j++)
{ if(sportifs1[i].IdSportif<sportifs2[j].IdSportif)
{ if(sportifs1[i].Age==sportifs2[j].Age)
{printjson('( '+sportifs1[i].IdSportif +' '+ sportifs1[i].Nom+'
'+sportifs1[i].Prenom+' , '+ sportifs2[j].IdSportif +'
'+sportifs2[j].Nom+' '+sportifs2[j].Prenom+' )); }}}}
```

## Exercice 02

n. Quelle est la moyenne d'âge des sportives qui pratiquent du basket ball ?

```
db.Sportifs.aggregate([
```

```
{ $match: { "Sports.Jouer": "Basket ball", "Sexe": { $in: [ "f", "F" ] } }},
```

```
{ $group: { "_id": null, "AgeMoyen": { $avg: "$Age" } } }  
])
```

## Exercice 02

o. Quels sont les sportifs les plus jeunes ?

□ On calcule l'âge min et on le stocke dans une variable `agemin`. Le calcul de l'âge min se fait en utilisant la fonction « aggregate ».

```
var agemin=db.Sportifs.aggregate(  
[{$group: {_id: null, "agemin": {$min:"$Age"}}}]).next();
```

□ On cherche les sportifs avec `Age = agemin.agemin`.

```
db.Sportifs.find({"Age": agemin.agemin, {"_id":0,  
"Sports":0})
```

## Exercice 02

o. **Quels sont les sportifs les plus jeunes ?**

□ On peut faire la requête en une seule ligne comme suit:

```
db.Sportifs.find({"Age":
```

```
db.Sportifs.aggregate([  
  {$group:      { _id:      null,      "agemin":  
    {$min: "$Age" }}}]).next().agemin},
```

```
  {"_id":0, "Sports":0}  
)
```

## Exercice 02

q. Quels entraîneurs n'entraînent que du hand ball ou du basket ball ?

❑ On récupère la liste des sports entraînés.

```
var sports = db.Sportifs.distinct("Sports.Entraîner");
```

❑ On fait un filtre pour supprimer le hand et le basket en utilisant la fonction **filter** .

```
var autres = sports.filter(function(s) {  
  return (s != "Hand ball" & s != "Basket ball")  
});
```

## Exercice 02

q. Quels entraîneurs n'entraînent que du hand ball ou du basket ball ?

□ On cherche ceux qui entraîne l'un ou l'autre, mais pas d'autres sports.

```
db.Sportifs.find(  
  {$and: [{ $or: [ { "Sports.Entraîner": "Hand ball" },  
                  { "Sports.Entraîner": "Basket ball" }  
                ] },  
          { "Sports.Entraîner" : { $nin : autres } } ]  
},  
{"_id": 0,  
 "Nom": 1,  
 "Sports.Entraîner": 1  
})
```

## Exercice 02

**t. Pour chaque sportif donner le nombre de sports qu'il arbitre**

```
db.Sportifs.aggregate([  
  { $match: { "Sports.Arbitrer": { $exists: true } }},  
  { $unwind: "$Sports.Arbitrer"},  
  { $group: { _id: "$Nom", "nbArbitrer": { $sum: 1 } } }  
])
```



## Exercice 02

**u. Pour chaque gymnase de Stains donner par jour d'ouverture les horaires des premières et dernières séances**

```
db.Gymnases.aggregate([
  { $match: { "Ville": "STAINS" } },
  { $unwind: "$Seances" },
  { $project: { "nom": "$NomGymnase", "jour":
    { $toLower: "$Seances.Jour" }, "h": "$Seances.Horaire" } },
  { $group: { _id: { "nom": "$nom", "jour": "$jour" },
    "debut": { $min: "$h" }, "fin": { $max: "$h" } } }
])
```

## Exercice 02

**u. Pour chaque entraîneurs de hand ball quel est le nombre de séances journalières qu'il assure ?**

❑ On récupère la liste des identifiants des entraîneurs de Hand.

```
var entraineursHand = db.Sportifs.find({ "Sports.Entraîneur" :  
"Hand ball" }, { _id: 0, IdSportif: 1 }).toArray().map(function(e) {  
return e.IdSportif });
```

❑ On les utilise pour faire la restriction (avec le \$match) dans l'agrégation.

```
db.Gymnases.aggregate([  
  { $unwind: "$Seances" },  
  { $match: { "Seances.IdSportifEntraîneur": { $in: entraineursHand  
  } }},  
  { $project : { "ent": "$Seances.IdSportifEntraîneur",  
    "jour": { $toLower: "$Seances.Jour" } }},  
  { $group: { _id: { "entraîneur": "$ent", "jour": "$jour"},  
    nbSeances: { $sum: 1 } } } ]
```

)

## Exercice 02

### 4. Le paradigme MapReduce en MongoDB

- ❑ MapReduce est un des mécanismes les plus complexes de requêtes de mongoDB.
- ❑ Il se base sur la spécification des deux fonctions Map et Reduce(écrites en javascript).
- ❑ Ces deux fonctions sont des fonctions définies par l'utilisateur.
- ❑ En mongoDB, la fonction Map, permet de générer des documents clés-valeurs pour les transmettre en entrée à Reduce. La fonction ne prend aucun paramètre, on accède à l'objet analysé via l'opérateur this. La fonction peut émettre des couples via la fonction emit(key, value) autant de fois que nécessaire dans la fonction.

## Exercice 02

### 4. Le paradigme MapReduce en MongoDB

□ La fonction Reduce retourne le résultat agrégé à partir de ces documents en entrée. La fonction prend deux paramètres key et values (tableau des valeurs de la clé). Elle peut être appelée plusieurs fois pour la même clé, elle doit donc renvoyer une valeur de même type que celles dans le tableau.

□ On doit passer un troisième paramètre, un littéral JSON, qui représente les options de la fonction. La principale option (out) est la collection dans laquelle le résultat sera placé. Si l'on veut voir le résultat, sans le stocker, il est possible d'indiquer out: { inline: 1 }

## Exercice 02

### a. Calculer le nombre de gymnases pour chaque ville

```
var Map=function() {  
    emit(this.Ville, 1); }
```

La fonction Map, crée un document avec comme clé la ville et 1 comme valeur. Ce document est transmis à la fonction Reduce

```
var Reduce=function(cle, nbgy) {  
    return Array.sum(nbgy); }
```

La fonction Reduce, crée un document clé/valeur avec comme clé ville et nbgy qui indique le nombre de gymnases pour cette ville

```
db.Gymnases.mapReduce( Map, Reduce,  
    { out: { inline: 1 } })
```

Exécution des fonctions Map et Reduce, sur la collection des gymnases et affichage des résultats

## Exercice 02

- a. Calculer le nombre de gymnases pour chaque ville (résultats):

```
/* 1 */
{
  "results" : [
    {
      "_id" : "GARGES",
      "value" : 1.0
    },
    {
      "_id" : "MONTMORENCY",
      "value" : 5.0
    },
    {
      "_id" : "PIERREFITTE",
      "value" : 5.0
    },
    {
      "_id" : "SAINT DENIS",
      "value" : 3.0
    },
    {
      "_id" : "SARCELLES",
      "value" : 5.0
    },
    {
      "_id" : "STAINS",
      "value" : 6.0
    },
    {
      "_id" : "VILLETANEUSE",
      "value" : 3.0
    }
  ],
  "timeMillis" : 77.0,
  "counts" : {
    "input" : 28,
    "emit" : 28,
    "reduce" : 6,
    "output" : 7
  },
  "ok" : 1.0,
}
```

## Exercice 02

### **b. Calculer le nombre de séances pour chaque jour de la semaine**

```
var Map=function() {  
  if (this.Seances) { // il y a des gymnases qui ne proposent pas de séances  
    for (s of this.Seances) // pour chaque item de Seances, émettre ses jours  
      { emit(s.Jour.toLowerCase(), 1);  
        } } };  
  
var Reduce =function(jours, nbseances) {  
  return Array.sum(nbseances); }  
  
db.Gymnases.mapReduce(Map, Reduce,  
  { out: { inline: 1 } } )
```

## Exercice 02

**d. Calculer la superficie moyenne des gymnases, pour chaque ville. Pour cela, vous devez calculer la somme des superficie ET le nombre de gymnase (à émettre dans un même objet et à réduire en tenant compte que ce double aspect)**

```
var Map=function() {  
  emit(this.Ville,  
  { "nb": 1,  
    "surface": this.Surface  
  });  
};
```

```
var Reduce=function(ville, valeurs) {  
  var nb = 0, surface = 0;  
  for (val of valeurs) {  
    nb += val.nb;  
    surface += val.surface;  
  }  
  return { "nb": nb,  
    "surface": surface,  
    "surfMoy": Math.round( surface / nb  
  ) };  
};
```

```
db.Gymnases.mapReduce(Map, Reduce, { out: { inline: 1 } })
```