UNIVERSITÉ PARIS DESCARTES | USPC Université Sorbonne Paris Cité	DUT STID **BASES NoSQL** TP Agrégation de données *F. Role*

Rappels

On définit un ***pipeline*** d'opérations (groupement, tri, etc.) qui s'applique à tous les documents d'une collection. Chaque opération est représentée par un document et le *pipeline* est matérialisé par une liste qui regroupe les documents représentant les opérations.

On crée ainsi une liste d'étapes de traitement. Chaque étape traite des documents reçus en entrée (au début tout ou partie d'une collection) et passe les documents traités à l'étape suivante.

La méthode `agregate(.)` permet d'appliquer le *pipeline*. On a donc une instruction de la forme :

```
db.mycoll.agregate( [{…}, {…}, {…}] )
```

```
Les principaux opérateurs sont : $group, $limit, $match, $poject, $sort, $unwind.
```

Exercice 1

Créez la collection villes en y insérant des enregistrements comme `{ ville : "paris, temperature : 10}` à partir de la liste suivante.

```
var villes=[["Paris" , 10 ],["Londres" , 12 ],["Rome", 16 ], ["Amsterdam" , 10 ],["Moscou" , 2 ] , ["Shangai" , 19 ], ["Tokyo" , 18 ],["Casablanca", 30 ],["Abidjan", 26 ], ["Rio", 26 ], ["Quito", 22 ], ["Mexico", 26 ] , ["Boston", 18 ] ]
```

Exercice 2

A partir du dictionnaire ci-dessous ajoutez à chaque enregistrement d'une ville de la collection villes l'indication du continent de cette ville.

```
var continents={"Paris" : "Europe" , "Londres" : "Europe" , "Rome" :"Europe" , "Amsterdam" : "Europe" , "Moscou" : "Europe" , "Shangai" : "Asie" , "Tokyo" : "Asie" , "Casablanca" : "Afrique" , "Abidjan": "Afrique" , "Rio": "Amerique" ,"Quito" :
```

```
"Amerique" , "Mexico" : "Amerique" , "Boston": "Amerique" } ;
```

Exercice 3

Même question que la précédente mais cette fois pour ajouter des informations de poulation à partir du dictionnaire

var populations={"Paris" : 5 , "Londres" : 7 , "Rome" :5 , "Amsterdam" : 2 , "Moscou" : 5 , "Shangai" : 15 , "Tokyo" :10, "Casablanca" : 3 , "Abidjan": 4 , "Rio": 9 ,"Quito" :2 , "Mexico" : 15 , "Boston": 5 }

A ce stade on a des enregistrements comme { "_id" : ObjectId("5a...e"), "ville" : "Mexico", "temperature" : 26, "population" : 15, "continent" : "Amerique" }

Exercice 4

Créez et testez la méthode aggregate() avec le code ci-dessous.

Attention aux " et $ qui sont nécessaires pour désigner les champs!!!

```
var cur = db.villes.aggregate({$group : {_id : "$continent"}}) ;
```

```
while( cur.hasNext()) printjson(cur.next()) ;
```

Exercice 5

Pour chaque continent, affichez son nombre de villes.

```
var cur = db.villes.aggregate([{$group : {_id : "$continent" , count : { $sum : 1} } } ] )
```

Exercice 6

Pour chaque continent affichez la température moyenne par ordre décroissant.

Pour cela, dans l'objet en valeur de $group on ajoute le couple attribut-valeur `moyenne : { $avg : "$temperature"}` et dans la liste, à la suite du l'élément `$group` on ajoute l'élément `{$sort : {moyenne : -1}}`

On doit avoir :

```
{ "_id" : "Afrique", "moyenne" : 28 }
{ "_id" : "Amerique", "moyenne" : 23 }
{ "_id" : "Asie", "moyenne" : 18.5 }
{ "_id" : "Europe", "moyenne" : 10 }
```

Exercice 7

Affichez a température moyenne des villes dont le nom commence par `A`.

Le premier élément de la liste doit être `{$match : {ville: /^A/} }`. On a ensuite un groupe avec `_id: null` pour dire qu'on ne partitionne pas.

Exercice 8 (révision)

Enlevez le champ `continent` de **tous** les enregistrements.

Exercice 1

Le thème general abordé dans cette séance d'exercices est la capacité à s'adapter à de nouvelles

Exercice 1

Le thème general abordé dans cette séance d'exercices est la capacité à s'adapter à de nouvelles

Exercice 1

Le thème general abordé dans cette séance d'exercices est la capacité à s'adapter à de nouvelles

Exercice 1

Le thème general abordé dans cette séance d'exercices est la capacité à s'adapter à de nouvelles

Exercice 1

Le thème general abordé dans cette séance d'exercices est la capacité à s'adapter à de nouvelles

Exercice 1

Le thème general abordé dans cette séance d'exercices est la capacité à s'adapter à de nouvelles

Exercice 1

Le thème general abordé dans cette séance d'exercices est la capacité à s'adapter à de nouvelles

Exercice 1

Le thème general abordé dans cette séance d'exercices est la capacité à s'adapter à de nouvelles

Exercice 1

Le thème general abordé dans cette séance d'exercices est la capacité à s'adapter à de nouvelles

Exercice 1

Le thème general abordé dans cette séance d'exercices est la capacité à s'adapter à de nouvelles

Exercice 1

Le thème general abordé dans cette séance d'exercices est la capacité à s'adapter à de nouvelles

Exercices

Le thème general abordé dans cette séance d'exercices est la capacité à s'adapter à de nouvelles demandes (changement du cahier des charges) ou à de nouvelles contraintes techniques. C'est l'essence même de l'esprit « Agile » qui incite à procéder par de courtes itérations de manière à pouvoir suivre au plus près les demandes du client (et leurs fluctuations…)

a) Enlevez tous les enregistrements meteo.

On repart de 0.

On enlève tous les enregistrements meteo.

```
db.meteo.remove({})  // plutôt que db.meteoo.remove() et pas db.meteo.drop()
```

b) Insérez le contenu de la liste Javascript villes = [[ ]] dans la collection meteo.

Nous avons une liste liste Javascript qui peut être insérée directement dans la base par une simple boucle.

```
> villes=[["Paris" , 10 ],["Londres" , 12 ],["Rome", 16 ], ["Amsterdam" , 10 ],["Moscou" , 2 ] , ["Shangai" , 19 ], ["Tokyo" , 18 ],["Casablanca", 30 ],["Abidjan", 26 ], ["Rio", 26 ], ["Quito", 22 ], ["Mexico", 26 ] , ["Boston", 18 ] ]
```

```
>  villes.forEach(function(e) {db.meteo.insert({"ville" : e[0] , "temperature" : e[1]}) } )
```

c) Récupérez le contenu de la liste de la collection meteo dans une liste Javacsript nommée temperatures

On récupère le contenu dans une liste.

```
> temperatures = db.meteo.find({}).map( function(e) {return e })
```

d) On a aussi un dictionnaire nommé continents qui donne pour chaque ville le continent où elle est située. Continets = {}.

Utilisez ce dictionnaire pour enrichir la collection meteo en ajoutant un champ continent à chaque enregistrement meteo. Pour chaque ville on connaîtra ainsi sa température mais aussi son continent.

On a aussi un dictionnaire qui donne pour chaque ville le continent où elle est située :

```
continents={"Paris" : "Europe" , "Londres" : "Europe" ,
"Rome" :"Europe" , "Amsterdam" : "Europe" , "Moscou" : "Europe"
, "Shangai" : "Asie" , "Tokyo" : "Asie" , "Casablanca" :
"Afrique" , "Abidjan": "Afrique" , "Rio": "Amerique" ,"Quito" :
"Amerique" , "Mexico" : "Amerique"   , "Boston": "Amerique"  }
```

On va s'en servir pour enrichir la collection `meteo` en ajoutant un champ `continent` à chaque enregistrement meteo. Pour chaque ville on connaîtra ainsi sa température mais aussi son continent.

```
> temperatures.forEach(function(t)
{ db.meteo.update( {"ville" : t.ville} ,  {"ville" : t.ville,
"temperature" : t.temperature , "continent" :
continents[t.ville]}, {upsert:true} )   })
```

db.meteo.find() doit renvoyer :

```
"_id" : ObjectId("558581db25c1600e88299da5"), "ville" :
"Paris", "temperature" : 10, "continent" : "Europe" }

"_id" : ObjectId("558581db25c1600e88299da6"), "ville" :
"Londres", "temperature" : 12, "continent" : "Europe" }

"_id" : ObjectId("558581db25c1600e88299da7"), "ville" : "Rome",
"temperature" : 16, "continent" : "Europe" }

"_id" : ObjectId("558581db25c1600e88299da8"), "ville" :
"Amsterdam", "temperature" : 10, "continent" : "Europe" }

"_id" : ObjectId("558581db25c1600e88299da9"), "ville" :
"Moscou", "temperature" : 2, "continent" : "Europe" }

"_id" : ObjectId("558581db25c1600e88299daa"), "ville" :
"Shangai", "temperature" : 19, "continent" : "Asie" }

"_id" : ObjectId("558581db25c1600e88299dab"), "ville" :
"Tokyo", "temperature" : 18, "continent" : "Asie" }
```

```
"_id" : ObjectId("558581db25c1600e88299dac"), "ville" :
"Casablanca", "temperature" : 30, "continent" : "Afrique" }

"_id" : ObjectId("558581db25c1600e88299dad"), "ville" :
"Abidjan", "temperature" : 26, "continent" : "Afrique" }

"_id" : ObjectId("558581db25c1600e88299dae"), "ville" : "Rio",
"temperature" : 26, "continent" : "Amerique" }

"_id" : ObjectId("558581db25c1600e88299daf"), "ville" :
"Quito", "temperature" : 22, "continent" : "Amerique" }

"_id" : ObjectId("558581d
```

Rappels

On définit un ***pipeline*** d'opérations (groupement, tri, etc.) qui s'applique à tous les documents d'une collection. Chaque opération est représentée par un document et le *pipeline* est matérialisé par une liste qui regroupe les documents représentant les opérations.

On crée ainsi une liste d'étapes de traitement. Chaque étape traite des documents reçus en entrée (au début tout ou partie d'une collection) et passe les documents traités à l'étape suivante.

La méthode `agregate(.)` permet d'appliquer le *pipeline*. On a donc une instruction de la forme :

```
db.mycoll.agregate( [{…}, {…}, {…}] )
```

Exercices

Le thème general abordé dans cette séance d'exercices est la capacité à s'adapter à de nouvelles demandes (changement du cahier des charges) ou à de nouvelles contraintes techniques. C'est l'essence même de l'esprit « Agile » qui incite à procéder par de courtes itérations de manière à pouvoir suivre au plus près les demandes du client (et leurs fluctuations…)

a) Enlevez tous les enregistrements meteo.

On repart de 0.

On enlève tous les enregistrements meteo.

```
db.meteo.remove({})  // plutôt que db.meteoo.remove() et pas
db.meteo.drop()
```

b) Insérez le contenu de la liste Javascript villes = [[ ]] dans la collection meteo.

Nous avons une liste liste Javascript qui peut être insérée directement dans la base par une simple boucle.

```
> villes=[["Paris" , 10 ],["Londres" , 12 ],["Rome", 16 ],
["Amsterdam" , 10 ],["Moscou" , 2 ] , ["Shangai" , 19 ],
["Tokyo" , 18 ],["Casablanca", 30 ],["Abidjan", 26 ], ["Rio",
26 ], ["Quito", 22 ], ["Mexico", 26 ] , ["Boston", 18 ] ]
```

```
>  villes.forEach(function(e) {db.meteo.insert({"ville" :
e[0] , "temperature" : e[1]}) } )
```

c) Récupérez le contenu de la liste de la collection meteo dans une liste Javacsript nommée temperatures

On récupère le contenu dans une liste.

```
> temperatures = db.meteo.find({}).map( function(e) {return
e })
```

d) On a aussi un dictionnaire nommé continents qui donne pour chaque ville le continent où elle est située. Continets = {}.

Utilisez ce dictionnaire pour enrichir la collection meteo en ajoutant un champ continent à chaque enregistrement meteo. Pour chaque ville on connaîtra ainsi sa température mais aussi son continent.

On a aussi un dictionnaire qui donne pour chaque ville le continent où elle est située :

```
continents={"Paris" : "Europe" , "Londres" : "Europe" ,
"Rome" :"Europe" , "Amsterdam" : "Europe" , "Moscou" : "Europe"
, "Shangai" : "Asie" , "Tokyo" : "Asie" , "Casablanca" :
"Afrique" , "Abidjan": "Afrique" , "Rio": "Amerique" ,"Quito" :
"Amerique" , "Mexico" : "Amerique"   , "Boston": "Amerique"  }
```

On va s'en servir pour enrichir la collection `meteo` en ajoutant un champ `continent` à chaque enregistrement meteo. Pour chaque ville on connaîtra ainsi sa température mais aussi son continent.

```
> temperatures.forEach(function(t)
{ db.meteo.update( {"ville" : t.ville} ,  {"ville" : t.ville,
"temperature" : t.temperature , "continent" :
continents[t.ville]}, {upsert:true} )   })
```

db.meteo.find() doit renvoyer :

```
"_id" : ObjectId("558581db25c1600e88299da5"), "ville" :
"Paris", "temperature" : 10, "continent" : "Europe" }

"_id" : ObjectId("558581db25c1600e88299da6"), "ville" :
"Londres", "temperature" : 12, "continent" : "Europe" }

"_id" : ObjectId("558581db25c1600e88299da7"), "ville" : "Rome",
"temperature" : 16, "continent" : "Europe" }

"_id" : ObjectId("558581db25c1600e88299da8"), "ville" :
"Amsterdam", "temperature" : 10, "continent" : "Europe" }

"_id" : ObjectId("558581db25c1600e88299da9"), "ville" :
"Moscou", "temperature" : 2, "continent" : "Europe" }

"_id" : ObjectId("558581db25c1600e88299daa"), "ville" :
"Shangai", "temperature" : 19, "continent" : "Asie" }

"_id" : ObjectId("558581db25c1600e88299dab"), "ville" :
"Tokyo", "temperature" : 18, "continent" : "Asie" }

"_id" : ObjectId("558581db25c1600e88299dac"), "ville" :
"Casablanca", "temperature" : 30, "continent" : "Afrique" }

"_id" : ObjectId("558581db25c1600e88299dad"), "ville" :
"Abidjan", "temperature" : 26, "continent" : "Afrique" }

"_id" : ObjectId("558581db25c1600e88299dae"), "ville" : "Rio",
"temperature" : 26, "continent" : "Amerique" }

"_id" : ObjectId("558581db25c1600e88299daf"), "ville" :
"Quito", "temperature" : 22, "continent" : "Amerique" }

"_id" : ObjectId("558581db25c1600e88299db0"), "ville" :
"Mexico", "temperature" : 26, "continent" : "Amerique" }

"_id" : ObjectId("558581db25c1600e88299db1"), "ville" :
"Boston", "temperature" : 18, "continent" : "Amerique" }
```

e) En utilisant l'opération group , regroupez les documents de la collection par continent.

```
> db.meteo.aggregate({$group : {_id : "$continent"}})

{ "_id" : "Amerique" }

{ "_id" : "Afrique" }
```

```
{ "_id" : "Asie" }
{ "_id" : "Europe" }
```

MongoDB a réparti les enregistrements dans différents groupes. Tous les enregistrements correspondant à un continent donné sont dans le même groupe.

f) En utilisant la fonction sum comptez le nombre de documents par continent.

On peut alors utiliser ces groupes comme avec une clause Group By en SQL. On peut par exemple compter le nombre d'enregistrements dans chaque bloc et voir que l'Europe est le continent le plus représenté dans la collection.

```
> db.meteo.aggregate({$group : {_id : "$continent" , count :
{ $sum : 1} }} )
{ "_id" : "Amerique", "count" : 4 }
{ "_id" : "Afrique", "count" : 2 }
{ "_id" : "Asie", "count" : 2 }
{ "_id" : "Europe", "count" : 5 }
```

g) Notre client nous informe qu'il est très intéressé par des statistiques prenant en compte le climat. Comme nous utilisons Mongo, nous sommes très agiles et pouvons nous adapter facilement à cette nouvelle demande. On nous envoie donc une liste qui donne le régime climatique (continental, tempéré, tropical) de chaque ville. Intégrez ces données.

```
> climat=[["Paris" , "tempere"],["Londres" , 12 ],["Rome",
16 ],["Amsterdam" , 10 ],["Moscou" , 2 ] , ["Shangai" , 19 ],
["Tokyo" , 18 ],["Casablanca", 30 ],["Abidjan", 26 ], ["Rio",
26 ], ["Quito", 22 ], ["Mexico", 26 ] , ["Boston", 18 ] ]
```

```
> db.meteo.find().forEach( function(t)  {
   for (var i = 0 ; i < climat.length ; i++ ) {
      if (climat[i][0] == t.ville ) {
          db.meteo.update( {ville : t.ville }, { ville :
t.ville , temperature : t.temperature , continent : t.continent
, climat : climat[i][1] } ,{upsert : false} )
```

} } })

h) Comme le client ne s'intéresse qu'aux villes des région tropicales, écrivez un pipline qui calcule pour chaque continent de la zone tropicale son nombre d'enregistrement et un pipline qui calcule pour chaque continent de la zone tropicale la température moyenne.

```
db.meteo.aggregate({$match : {"climat" : "tropical" }  } ,
{$group : {_id : "$continent" , count : { $sum : 1} }} )
```

```
db.meteo.aggregate({$match : {"climat" : "tropical" }  } ,
{$group : {_id : "$continent" , temp_moyenne : {$avg :
"$temperature"}  }}
```

i) Ecrivez les requêtes SQL correspondabt aux deux pipelines précédents.

j) Triez les villes par ordre décroissnat de température.

```
{ $sort :  {"temp_moyenne" : -1 }}
```