



Travaux Pratiques

ESILV

nicolas.travers (at) devinci.fr

1	Interrogation simple	3
1.1	Mise en place	3
1.1.1	Installation	3
1.1.2	Visualisation du contenu	3
1.1.3	Schéma	3
1.2	Requêtes <i>find</i>	4
1.3	Requêtes "distinct"	5
1.4	API Mongo - Requêtes <i>aggregate</i>	5
2	Mises à jour	6
2.1	Insertion	6
2.2	Update/Delete	6
2.3	Exercices	7
3	Indexation, 2DSphere & Lookup	8
3.1	Indexation simple (type scalaire)	8
3.2	Indexation 2D avec 2DSphere	9
3.3	\$Lookup	9
4	Map/Reduce	11
4.1	Map	11
4.2	Reduce	11

1.1 Mise en place

1.1.1 Installation

Pour l'installation de l'environnement MongoDB, installation du serveur, de la base de données/collections, le logiciel Studio3T, et les instructions sur le langage, veuillez vous référer au guide dans l'Annexe ??.

- Télécharger le jeu de données "tourPedia" disponible sur la plateforme ou ici : https://chewbii.com/tour-pedia_paris-json-2/
- Décompresser l'archive
- Importer le fichier dans une base "tourPedia" et une collection "paris"

1.1.2 Visualisation du contenu

Après vous êtes connecté sur Studio3T (cf Guide - Annexe ??), regarder à quoi ressemble un document avec le mode Collection, avec une requête vide.


Nous appellerons "*lieux*" les documents présents dans cette collection. Dont voici une extraction :

```
{
  "_id" : 455674,
  "name" : "Bibliothèque du Cnam",
  "category" : "poi",
  "location" : {
    "coord" : { "coordinates" : [2.354878, 48.866599], "type" : "Point" },
    "address" : "292 Rue Saint-Martin, Paris, France",
    "city" : "Paris"
  },
  "reviews" : [ ],
  "contact" : {
    "website" : "http://bibliotheque.cnam.fr",
    "GooglePlaces" : "https://plus.google.com/103448496999303589086/about?hl=en-US",
    "phone" : "+33 1 40 27 27 03",
    "foursquare" : "",
    "Booking" : "",
    "Facebook" : ""
  },
  "description" : "",
  "services" : [ ]
}
```

Cette collection contient des lieux de Paris qui ont été agrégé sur le site tourPedia. Vous y trouverez différents catégories, et informations pour les lieux :

- Des POI (points d'intérêts) ;
- Des restaurants ;
- Des logements (accommodation), avec les services associés ;
- Des attractions ;
- A chaque lieu sera associé des commentaires sur internet (Facebook, Foursquare), des notes utilisateurs.

1.1.3 Schéma

Studio3T offre la possibilité (sous licence) de consulter les statistiques d'apparition des clés et valeurs dans la collection. Pour cela, sélectionnez la collection cible, cliquez sur l'icone **schema** . Vous pouvez choisir d'effectuer

l'analyse du schéma sur l'ensemble de la base (traitement long) ou un extrait (moins précis). La figure ci-dessous illustre l'extraction de statistiques d'apparition de tous les documents de la base.

Fields	Global Probability	Type
▼ [tourPedia.paris]	100.0%	Collection
▶ _id	100.0%	...
▼ category	100.0%	String
"-" String	100.0%	String
▼ contact	100.0%	Object
▼ {} Object	100.0%	Object
▶ Booking	0.0%	String
▶ Facebook	7.7%	String
▶ Foursquare	8.2%	String
▶ GooglePlaces	92.3%	...
▶ phone	92.4%	...
▶ website	30.6%	String
▼ description	100.0%	...
□ Null	85.4%	Null
"-" String	14.6%	String
▼ likes	100.0%	Object
▼ {} Object	100.0%	Object
▶ Facebook	7.7%	...
▶ Foursquare	7.7%	...
▼ location	100.0%	Object
▼ {} Object	100.0%	Object
▶ address	91.3%	String
▶ city	92.3%	String
▶ coord	100.0%	Object

FIGURE 1.1 – Extrait de statistiques sur le schéma de la collection "Paris"

Pour effectuer les **requêtes** sur votre base de données, un guide rapide sur le **langage** est fourni dans le guide à la fin du document (Annexe ??).

1.2 Requêtes *find*

Créer des requêtes simples pour les phrases suivantes :

Donner les requêtes sur la collection paris répondant aux requêtes suivantes :

- 1.2.1 Donner le nom des lieux dont la catégorie est "accommodation"
- 1.2.2 Donner le nom et numéro de téléphone des lieux ayant un numéro de téléphone renseigné (\$exists, \$ne) ;
- 1.2.3 Nom et contacts des lieux ayant "website" et "Foursquare" renseignés ;
- 1.2.4 Nom des lieux dont le nom contient le mot "hotel" (attention à la casse) ;
- 1.2.5 Nom et services des lieux ayant un service "chambres non-fumeurs" ;
- 1.2.6 Nom et services des lieux dont le premier service est "chambres non-fumeurs" ;
- 1.2.7 Nom et services des lieux n'ayant qu'un seul service "chambres non-fumeurs" ;
- 1.2.8 Nom et services des lieux proposant 5 services ;
- 1.2.9 Nom et services des lieux proposant au moins 5 services ;
- 1.2.10 Donner les adresses des lieux de catégorie "accommodation" avec un service "blanchisserie" ;
- 1.2.11 Catégories des lieux ayant au moins une note (reviews.rating) de 4 ou plus ;
- 1.2.12 Affiche les sources des commentaires des lieux avec au moins un commentaire "Facebook" (source) ;
- 1.2.13 Donner les noms et notes de lieux ayant au moins une note de 4 ou plus, mais aucune note inférieure à 4 ;
- 1.2.14 Donner la langue et note des commentaires (reviews) de lieux, contenant au moins un commentaire écrit en anglais avec une note supérieure à 3 (attention, LE commentaire en anglais doit avoir un rang de 3 et plus).
Pour filtrer deux critères en même temps sur chaque élément d'une liste : \$elemMatch : {...}
- 1.2.15 Coordonnées GPS des lieux dont l'adresse contient "rue de rome"

1.3 Requêtes "distinct"

1.3.1 Liste distincte des catégories

1.3.2 Liste distincte des services

1.3.3 Liste distincte des sources pour les reviews

1.4 API Mongo - Requêtes *aggregate*

L'opérateur **aggregate** est une séquence d'opérations représentant une chaîne de pipeline entre chaque opérateur (le résultat d'un opérateur est donné à l'opérateur suivant). Il s'exprime de la sorte :
`aggregate([{$op1}, {$op2}, {$op3}])`.

- 1.4.1 Donner les adresses des lieux de catégorie "accommodation" avec un service "*blanchisserie*", projeter le résultat sur le nom et numéro de téléphone (seulement si elle existe), et trier sur le nom ;
- 1.4.2 Nombre de lieux de catégorie "accommodation" et ayant un service "*chambres non-fumeurs*" ;
- 1.4.3 Donner le nombre de lieux par catégorie ;
- 1.4.4 Pour les lieux de catégorie "accommodation", donner le nombre de lieux pour chaque service ;
- 1.4.5 Trier le résultat précédent par ordre décroissant ;
- 1.4.6 Du résultat précédent, n'afficher que les services présents dans plus de 1000 lieux ;
- 1.4.7 Pour chaque nom de lieu de catégorie "poi", donner le nombre de commentaires dont la source (reviews.source) est "Facebook". Trier par ordre décroissant ;
- 1.4.8 Pour chaque langue d'un commentaire (reviews.language), donner le nombre de commentaires de lieux ayant un service "*chambres non-fumeurs*" ;
- 1.4.9 Pour chaque nom de lieu de catégorie "restaurant", donner la note moyenne et le nombre de commentaires. Trier le résultat par ordre décroissant de moyenne, puis de nombre ;
- 1.4.10 Pour chaque catégorie de lieux et langue de commentaire, donner le nombre de commentaires correspondants ;
- 1.4.11 Pour chaque catégorie de lieux, donner le nombre moyen de commentaires par langue (réutiliser le résultat précédent) ;
- 1.4.12 Donner le nombre de moyen de commentaires par lieu ;

2.1 Insertion

- Créer un document (changer les valeurs qui vous concerne dans un éditeur) :

```
lieu = {
  "_id" : 1234567890,
  "name" : "METTRE LE NOM DE VOTRE ETABLISSEMENT",
  "reviews" : [{ "wordsCount" : 32, "rating" : 5, "language" : "fr",
    "text" : "LE TEXTE QUE VOUS POUVEZ METTRE ICI" } ],
  "contact" : {
    "website" : "URL DE VOTRE ETABLISSEMENT",
    "GooglePlaces" : "IDEM",
    "phone" : "CONTACT",
    "foursquare" : "URL FOURSQUARE",
    "Booking" : "",
    "Facebook" : "PAGE FACEBOOK"
  },
  "description" : {
    "fr" : "DESCRIPTION DE VOTRE ETABLISSEMENT",
    "de" : "", "en" : "", "it" : "", "nl" : "", "es" : ""
  },
  "location" : {
    "coord" : { "coordinates" : [LATITUDE, LONGITUDE], "type" : "Point" },
    "address" : "ADRESSE",
    "city" : "VILLE"
  },
  "services" : [ ], "category" : "poi"
};
```

- Insérer le document dans la collection “paris” :

```
db.paris.save(lieu);
```

- Consulter le résultat :

```
db.paris.find({"_id" : 1234567890}).pretty();
```

2.2 Update/Delete

Une fois la base intégralement téléchargée, vous pourrez faire des mises à jour, avec les instructions suivantes :

```
db.paris.update (
  {"_id" : 1234567890},
  {$set : { "type" : "Etablissement" } }
);
//Premier JSon : Mapping, Second JSon : Update ($set, $unset)
```

```
db.paris.find({"type":{"$exists:1}});
//Vérifier quels documents ont été modifiés
```

```
db.paris.remove({"_id" : 1234567890});
//Mapping de suppression
```

```
db.paris.find({"type":{"$exists:1}});
//Vérifier si les documents existent encore
```

- Il est possible de faire des fonctions itératives (javascript) sur le résultat du find :

```
db.paris.find( {"reviews.1":{"$exists:1}} ).forEach(
  function(lieu){
    lieu.nb_reviews = lieu.reviews.length;
    db.paris.save(lieu);
  }
);
//cela peut prendre un peu de temps pour calculer cela sur chaque document.
```

- Tester le résultat :

```
db.paris.aggregate([
  {$group : { _id : "$nb_reviews", "nb" : {$sum :1}}},
  {$sort : {"nb" : -1}}
]);

>>>>>
{ "_id" : null, "nb" : 45785 }
{ "_id" : 5, "nb" : 3145 }
{ "_id" : 2, "nb" : 2395 }
{ "_id" : 3, "nb" : 1541 }
{ "_id" : 4, "nb" : 1032 }
{ "_id" : 6, "nb" : 203 }
{ "_id" : 7, "nb" : 171 }
{ "_id" : 8, "nb" : 169 }
{ "_id" : 9, "nb" : 148 }
{ "_id" : 13, "nb" : 133 }
{ "_id" : 12, "nb" : 131 }
{ "_id" : 10, "nb" : 119 }
//45785 documents n'ont pas de reviews (pas de mises à jour effectuées),
//la plupart des lieux ont entre 4 et 5 commentaires.
```

2.3 Exercices

Pour les mises à jour suivantes, vérifier le contenu des données avant et après la mise à jour.

- 2.3.1 Mettre à jour tous les noms de lieux contenant “montparnasse” en ajoutant l’attribut “area” : “Montparnasse” (pour plusieurs mises à jour, l’option “multi :true” est à utiliser) ;
- 2.3.2 Supprimer la clé “nb_reviews” de tous les articles ;
- 2.3.3 Supprimer tous les lieux n’ayant aucun commentaire ;
- 2.3.4 Modifier tous les lieux pour ajouter un champ “wordsCount” qui correspond à la somme du nombre de mots de tous les commentaires (reviews.wordsCount)

Il est possible d'indexer des clés sous MongoDB. Nous allons voir comment les créer, et comment constater les modifications sur les plans d'exécution (`explain()` à partir de la version 2.6).

3.1 Indexation simple (type scalaire)

3.1.1 Pour la requête ci-dessous, regarder le plan d'exécution généré avec `“explain()”` à la fin de la requête :

```
db.paris.find({"services" : "chambres non-fumeurs", "reviews.rating" : {$gte : 4}}).explain();
```

3.1.2 On remarquera qu'une opération de "type" COLSCAN est appliqué (WinningPlan). Cela correspond à un parcours intégral de la collection.

3.1.3 Créer un index sur l'attribut année `“ db.paris.createIndex({"services":1});”`;

3.1.4 Exécuter à nouveau la requête en regardant le plan d'exécution généré;

3.1.5 Nous pourrions alors constater qu'une opération IXSCAN est appliquée sur la clé "services". Le nouvel index est utilisé. Créer un index sur la clé "reviews.rating". Vous pourrez alors constater que le plan d'exécution est identique, toutefois, un plan d'exécution a été rejeté "rejectedPlans" (celui sur le "reviews.rating")

3.1.6 Il est également possible de consulter le plan d'exécution pour les agrégats avec l'option (2° paramètre de "aggregate") `{explain:true}` :

```
db.paris.aggregate([{$match:{"services" : "chambres non-fumeurs"}},
  {$group:{$_id:"$type", total : { $sum : 1}}}], {explain:true});
```

3.1.7 Pour MapReduce, il n'est pas possible de consulter le plan d'exécution. Toutefois, on peut constater l'utilisation de l'index via le nombre d'éléments en 'input' :

```
var mapFunction = function () {
  for(var i=0 ; i < this.reviews.length ; i++){
    if(this.reviews[i].rating > 4)
      emit(this.reviews[i].language, 1);
  }
};
var reduceFunction = function (key, values) {
  return Array.sum(values);};
var queryParam = {"query":{}};
db.paris.mapReduce(mapFunction, reduceFunction, queryParam);
```

3.1.8 Vous pourrez constater que l'ensemble des documents sont interrogés (input :15439). En effet, le prédicat "reviews" est présent dans le map. L'optimiseur de MongoDB ne peut analyser la fonction map (appliquée à chaque document), il faut pour cela utiliser le paramètre "query" du queryParam pour que l'index soit pris en compte :

```
var mapFunction = function () {
  for(var i=0 ; i < this.reviews.length ; i++){
    if(this.reviews[i].rating > 4)
      emit(this.reviews[i].language, 1);
  }
};
var reduceFunction = function (key, values) {
  return Array.sum(values);};
var queryParam = {"query":{"reviews.rating" : {$gt : 4}}};
db.paris.mapReduce(mapFunction, reduceFunction, queryParam);
```

Seul 7909 documents sont interrogés, ceux qui nous intéressent.

3.2 Indexation 2D avec 2DSphere

Nous pouvons indexer les données avec 2DSphere qui permet de faire des recherches en 2 dimensions. Le schéma des coordonnées doit être structuré de la manière suivante¹ :

```
"location" : {
  "coord" : {
    "type" : "Point",
    "coordinates" : [
      1.53414,
      42.50729
    ]
  }
}
```

La clé de localisation dans ce document est "location.coord".

L'attribut location sera alors indexé :

```
db.paris.ensureIndex( { "location.coord" : "2dsphere" } );
```

Pour interroger l'index, il faut utiliser un opérateur 2D et l'utiliser sur "location.coord"
Documentation : <http://docs.mongodb.org/manual/tutorial/query-a-2d-index/>.

3.2.1 Tout d'abord récupérer les coordonnées de :

- "Eiffel Tower Paris France"
- "Pyramide du Louvre"
- "Boulevard Saint-Michel"

Affecter des variables tourEiffel, louvre et saintMichel à ces coordonnées pour les réutiliser par ailleurs ;

3.2.2 Afficher les noms et adresses des restaurants autour de saint-Michel dans un rayon de 200m. Pour cela, utiliser la syntaxe suivant :

```
{"location.coord":{"$near":{"$geometry":{"type":"Point","coordinates":[LAT, LONG]}, "$maxDistance":XXX}}};
```

(distance en mètres)

3.2.3 Calculer la note moyenne des restaurants de cette zone. Dans un **aggregate** il faut utiliser l'opérateur **\$geoNear** :

```
{"$geoNear": {"near":{"type":"Point", "coordinates" : [ LAT, LONG ]},"maxDistance":XXX,
"distanceField" : "outputDistance", "spherical":true}};
```

Par ailleurs il faut activer la version "3.4" : `db.adminCommand({ setFeatureCompatibilityVersion: "3.4" })`

3.2.4 idem avec la Tour Eiffel et le Louvre

3.2.5 Afficher le nom des points d'intérêts (poi) compris dans le triangle "tour Eiffel - Louvre - Saint-Michel" avec l'opérateur **\$geoWithin** :

```
{"$geoWithin":{"$geometry":{"type":"Polygon","coordinates": [triangle]}}}
```

Le triangle est une liste de points avec retour au point de départ. La requête est une liste de polygones.

3.2.6 Calculer le nombre de lieux par catégorie dans cette zone ;

3.3 \$Lookup

Cette fonctionnalité est un opérateur "aggregate" disponible depuis la version 3.4, permet de faire des jointures "gauches". C'est à dire que pour chaque document *source* il vérifie l'existence de la clé de jointure dans une collection externe. Si c'est le cas, il imbrique le(s) document(s) correspondant(s) dans une liste imbriquée dans le document source, sinon, le document n'est pas modifié (ni filtré).

Il faut pour cela préciser la clé à filtrer, puis la clé à vérifier dans la collection externe :

1. Documentation : <http://docs.mongodb.org/manual/applications/geospatial-indexes/>

```
{
  $lookup: {
    "from": "<collection destination>",
    "localField": "<clé à filtrer>",
    "foreignField": "<clé à aller vérifier (provenant du from)>",
    "as": "<nom de la clé contenant la liste des documents joints>"
  }
}
```

Pour illustrer cela, nous allons créer une collection de données personnalisées. Nous allons y stocker tous vos commentaires sur les destinations que vous avez visité. Pour cela, il faudra y associer l'identifiant du lieu que vous commentez. Exemple :

```
db.comments.save({
  "user":1, "idLocation": 231445, "date":"2017-02-03",
  "comment" : "Impressive place, especially at sunset"
});
```

3.3.1 Créer 5 documents pour la collection "comments" sur 5 lieux différents (ou non).

3.3.2 Testons la jointure :

```
db.paris.aggregate([
  {$lookup : {
    "from" : "comments",
    "localField":"_id",
    "foreignField" : "idLocation",
    "as" : "my_comments"
  }}
]);
```

3.3.3 Quelle est la taille du résultat produit ?

3.3.4 Quelle est la taille du résultat après filtrage sur l'existence de la clé "my_comments" ?

3.3.5 Pour être plus efficace dans cette opération, il est recommandé de créer un index ou de "sharder" la collection "comments" sur la clé "idLocation". Créer un index et tester à nouveau la requête.

Dans ce chapitre, nous allons étudier le comportement d'une requête Map/Reduce sous MongoDB. Pour ce faire, nous allons commencer par s'intéresser au Map, puis à la partie Reduce.

4.1 Map

Cette section s'intéresse aux variantes sur la partie Map de la requête.

4.1.1 Exécuter la requête suivante :

```
mapFunction = function () {  
    emit(this.category, 1);  
};  
reduceFunction = function (key, values) {  
    return Array.sum(values);  
};  
queryParam = {"query": {}, "out": {"inline": true}};  
db.paris.mapReduce(mapFunction, reduceFunction, queryParam);
```

- 4.1.2 Pour chaque catégorie, donner le nombre de commentaires en français (utiliser la taille d'une liste en javascript "reviews.language");
- 4.1.3 Pour chaque catégorie, donner le nombre de mots pour les commentaires en français;
- 4.1.4 Pour chaque "type" de contact (s'il existe), donner le nombre de lieux associés;
- 4.1.5 Donner le nombre de lieux pour chaque "nombre moyen de mots" par lieu (moyenne de reviews.wordsCount), pour les messages Facebook. Arrondir la moyenne à l'inférieur (Math.floor());
- 4.1.6 Compter le nombre de lieux ayant plus de 3 commentaires en anglais;
- 4.1.7 Compter par langue, le nombre de lieux ayant plus de 3 commentaires de cette langue;
- 4.1.8 Compter le nombre du lieux par nombre de services ;
- 4.1.9 Pour chaque nom de lieu, donner ceux qui sont présents au moins deux fois.
Attention, la fonction reduce n'est évaluée que lorsqu'il y a au moins 2 documents pour une même clé. Il est donc nécessaire d'appliquer un filtre après génération du résultat (stocker le résultat pour faire une requête dessus). Trier le résultat par ordre décroissant du nombre d'occurrences;
- 4.1.10 Pour la catégorie "accommodation", donner le nombre de commentaires par année (utiliser 'substring' sur la date);
- 4.1.11 Pour les commentaires Foursquare, donner le nombre de commentaires par année;
- 4.1.12 Pour chaque clé "source & année", donner le nombre de publications.
Attention, la clé du emit doit être un document ;
- 4.1.13 Pour chaque couple de service, donner le nombre de lieux;

4.2 Reduce

Dans cette section, nous allons modifier la fonction **reduce**.

- 4.2.1 Pour chaque source de commentaire, donner le nombre moyen de mots;
Attention, la moyenne n'est pas une fonction associative (contrairement à la somme) - problème de reduce local et global.
- 4.2.2 Même question, mais en produisant la note moyenne (rating), min et max;
- 4.2.3 Donner la liste distincte des services proposés;
- 4.2.4 Pour chaque source, nombre de langues de commentaires différents par type de source;
- 4.2.5 Pour chaque source de commentaire, donner la moyenne du nombre de commentaires par année. Réutiliser la requête "source&annee", puis calculer la moyenne avec "aggregate";
- 4.2.6 Le refaire, mais en une seule requête "map/reduce";