**Comparaison avec et sans index sur des recherches.**

Contexte : Comparaison de l’utilisation d’index sur la performance de requêtes plus ou moins complexe sur une collection de 1000 documents.

Voici un example de requête utilisée :

*db.runCommand(*

*{*

*explain: { count: « livres », query: { find: { auteur: « Auteur 50 » }}},  
 verbosity : « executionStats »*

*}*

*)*

Après avoir exécuté une requête sans index pour chaque comparaison à réaliser nous créons des index simple ou composite suivant la requête.

Index simple :

*db.livres.createIndex(*

*{ auteur: 1 },*

*{*

*background: true,*

*unique: true,*

*sparse: false,*

*name: "idx\_prix\_note\_moyenne"*

*}*

*)*

Index composite :

*db.livres.createIndex(*

*{ prix: 1, note\_moyenne: 1 },*

*{*

*background: true,*

*unique: true,*

*sparse: false,*

*name: "idx\_prix\_note\_moyenne"*

*}*

*)*

Recherche par titre exact

|  |  |  |
| --- | --- | --- |
|  | Avant index | Avec index |
| Nombre de doc examinés | 1000 | 1000 |
| Temps exécution (millisec) | 0 | 0 |
| Type d’étape utilisée | COLLSCAN | IXSCAN |

Recherche par auteur exact

|  |  |  |
| --- | --- | --- |
|  | Avant index | Avec index |
| Nombre de doc examinés | 1000 | 1000 |
| Temps exécution (millisec) | 0 | 0 |
| Type d’étape utilisée | COLLSCAN | IXSCAN |

Recherche par plage de prix

|  |  |  |
| --- | --- | --- |
|  | Avant index | Avec index |
| Nombre de doc examinés | 1000 | 1000 |
| Temps exécution (millisec) | 0 | 0 |
| Type d’étape utilisée | COLLSCAN | IXSCAN |

Recherche par langue et genre et tri par titre

|  |  |  |
| --- | --- | --- |
|  | Avant index | Avec index |
| Nombre de doc examinés | 1000 | 1000 |
| Temps exécution (millisec) | 0 | 0 |
| Type d’étape utilisée | COLLSCAN | IXSCAN |

Interprétation :

Il est malheureusement compliqué de pouvoir comparer sur 1000 documents au sein de la collection car celle-ci ne semble pas assez importante afin de réaliser une comparaison efficace entre l’absence et la présence d’index (nous pourrions soit augmenter considérablement le volume de data ou limité la RAM de notre machine afin de simuler des ressources plus limitées). En revanche, nous pouvons noter un changement du type d’étape utilisée qui met en évidence l’utilisation des index (IXSCAN).

Index spécialisé :

|  |  |  |
| --- | --- | --- |
|  | Avant index texte | Avec index texte |
| Nombre de doc examinés | 1000 | 1000 |
| Temps exécution (millisec) | 0 | 0 |
| Type d’étape utilisée | COLLSCAN | COLLSCAN |

En utilisant une requête via un find la méthode de recherche ne change pas (COLLSCAN), afin d’activé l’index de type texte il faut utiliser l’opérateur $search. Cependant en activant le $search on rédige la requête suivante :

*db.livres.find({ $text : { $search : « Titre 66 » } })*

Cela complexifie alors la requête et augmente le nombre d’étape (FETCH >TEXT\_MATCH > IXSCAN ). On observe donc plus d’étape qu’avec une simple requête find et un temps d’exécution plus long (2 millisecondes contre 0 avec un find sans index).

Index TTL :

*db.sessions\_utilisateurs.createIndex({ derniere\_connection: 1, expireAfterMinutes: 30 })*

Amélioration d’expérience :

Afin d’éviter la répétition de code (utilisation des requêtes permettant la création des index), j’ai créé des fonctions permettant de créer ces index.

Création d’index simple :

*function createSimpleIndexLivres(attr) {*

*db.livres.createIndex(*

*{ `${attr}`: 1 }*

*)*

*}*

Création d’index composite :

*function createCompositeIndexLivres(attributes) {*

*const object = {}*

*attributes.forEach((attribute) => object[attribute] = 1)*

*db.livres.createIndex(*

*object*

*)*

*}*

Ensuite je crée une une fonction sélectionnant pour moi la fonction a utilisé suivant si je passe plusieurs champs à mettre un index et donc créer un index simple ou un index composite.

*function createIndexLivres(param) {*

*if (Array.isArray(param)) {*

*createCompositeIndexLivres(param);*

*} else {*

*createSimpleIndexLivres(param);*

*}*

*}*

Création d’un index composite pour tester une requête couverte.

Utilisation de ma fonction : *createIndexLivres([‘titre’, ‘auteur’, ‘genres’, ‘langues’])*

Puis appel de la requête :

*db.livres.find({ titre: "Titre 1001", auteur: "Auteur 255", langues: "Espagnol", genres: "Drame"}).explain("executionStats")*

Temps d’exécution 4 millisecondes, utilisation de différentes étapes mais dès la première on voit que les étapes suivants la première (IXSCAN) sont mises en rejet. Probablement car dès la première étape la requête ne renvoie pas de document correspondant à la requête.

Index simple sur ISBN :

Après avoir créé un index simple sur ISBN et tester la création d’un document ayant le même ISBN qu’un autre pas de problème lors de la création. En revanche si on passe l’option unique à true lors de la création de l’index alors ce « duplicat » d’ISBN sera impossible.

Index partiel :

Amélioration des fonctions de créations des index afin de passer les options en second paramètre, puis création de l’index via la fonction pour un index partiel :

*createIndexLivres(‘titre’, {partialFilterExpression : {stock : {$gt : 0} } })*

Profiler les requêtes lentes :

Impossibilité sans création d’une base de donnée locale ou via docker de réaliser cette étape.

Suppression d’un index :

Via la requête :

*db.livres.dropIndex(« titre\_1 »)*

Les index le plus performants :

Les index simples sont les plus performants.

Dans une application de bibliothèque en production je privilégierai les index simple et composite sur des filtres à multi critères.   
En revanche si le client demande des systèmes de search dynamique via un champ alors il faudrait mettre en place un index texte en appliquant à cette fonctionnalité une méthode de requête via $search.

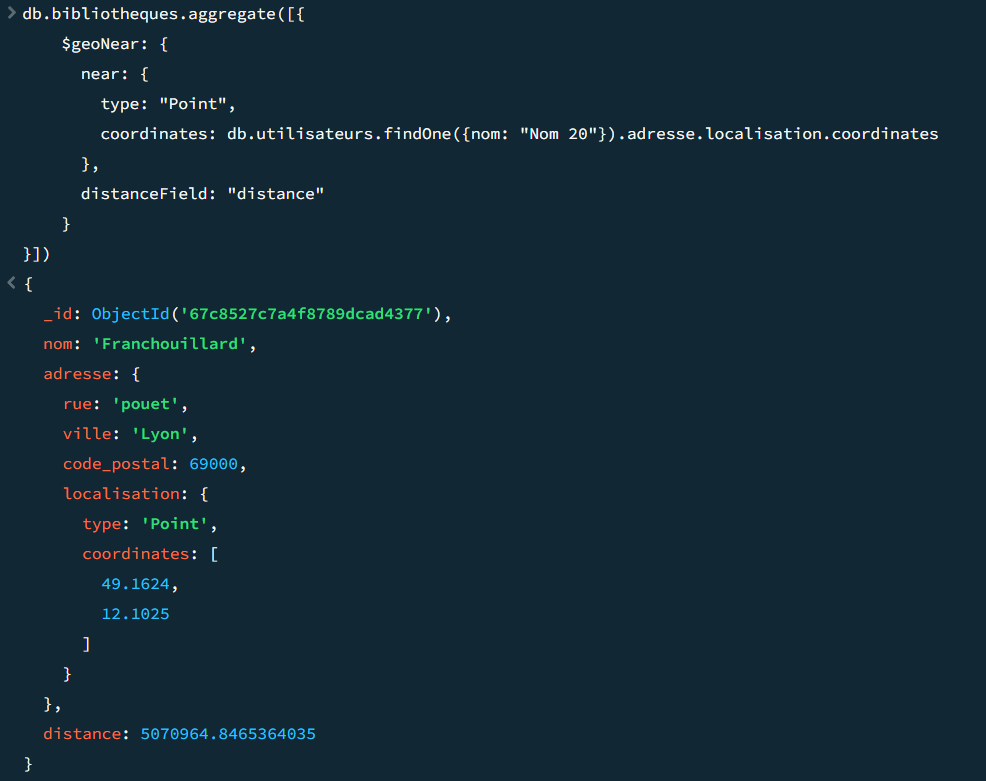
**Requêtes géospatiales**

Trouver le plus proche de paris :

Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.  
Je n’ai personne dans les 5km les plus proches c’est pourquoi j’utilise findOne afin de trouver le premier plus proche de paris.

Trouvez les bibliothèques les plus proches d’un user donné (trié par le plus proche) :



Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.



Pour obtenir la distance en km je passe en option distanceMultiplier avec la valeur 0.001 :

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Trouver les user dans un rayon d’une position (767km) :   
Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.

Trouver les utilisateurs dans la zone d’activité d’une bibliothèque :   
Afin de déterminer une zone d’activité « pertinente » pour les bibliothèques, j’ai voulu savoir la distance la plus faible séparant les utilisateurs des bibliothèques. J’ai donc saisie la requête suivante pour chaque bibliothèque :   
Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.

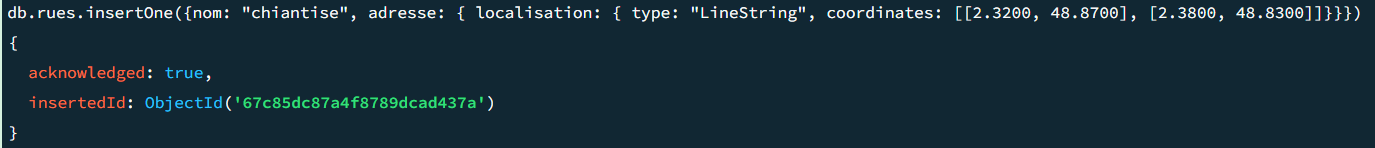
Je récupère ensuite le premier document listé. La distance la plus faible est à 5070 km environ.   
  
Je vais donc définir une distance d’activité de 6000 km pour avoir des résultats dans ma requêtes.

Une image contenant texte, capture d’écran

Le contenu généré par l’IA peut être incorrect.

Trouver si une bibliothèque se trouve dans une rue :

Définir la rue dans la collection rues :



Réaliser la requête pour trouver si une bibliothèque se situe dans cette rue :   
Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Il n’ y a donc aucune bibliothèque dans cette rue.

Application métier :

Création d’une livraison :

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Création d’une fonction pour update la position actuelle du livreur :   
Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Trouver toutes les livraisons comprises dans un rayon d’1 km d’un point données :   
Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Les données de géolocalisation permettraient, dans un service de bibliothèque, de connaitre la localisation des livres en temps réels, de la livraison à la réception. Cela permet de connaitre et l’étape de la vente ou location du livre ainsi que si celui-ci a bien été réceptionné par le client.

Dans un système comme celui-ci, il est impératif de maitriser la création des index géospatiale de notre collection.

Enfin, je réaliserai une intégration de ce type de fonctionnalité en la couplant a une API telle que google maps afin de suivre en temps réel les livraisons et d’améliorer la traçabilité de mes produits ainsi que de mes livraisons.