**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 })*

Juste pour améliorer l’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);*

*}*

*}*