Compte-Rendu NoSQL

Inaki Urrutia

1 Save/Insert

Q. Quelle est la différence entre la méthode insert et la méthode save ? Votre explication devra être illustrée en utilisant un jeu de test adapté à votre base.

R. save() utilise la fonction update() si le _id est fourni, sinon il utilise insert()

Exemple:

db.magasin.save(" _id ": ObjectId(_id existant), " libellé ": " chausettes ")
WriteResult(" nMatched " : 1, " nUpserted " : 0, " nModified " : 1)
– Modifie l'objet avec le _id correspondant

db.magasin.save(" id ": 6, " libellé ": " chausettes ")
WriteResult(" nInserted " : 1)
– Insere un nouvel objet car _id non spécifié

2 Requetes

Q. Obtenir les restaurants grecques du Queens ayant eu une note inférieure à 6 et pas de notes supérieures à 15. Vous afficherez le nom et la liste des scores.

R. db.getCollection('restaurants').find(
{"borough":"Queens",
"cuisine":"Greek",
$and: [{"grades.score": {$lt : 6}}, {"grades.score": {$not:{$gt : 15}}}]},
{"_id" : 0, "name" : 1, "grades.score" : 1})

Q. Obtenir les restaurants grecques du Queens ayant eu une note inférieure à 35 avec une évaluation C. Vous afficherez le nom et la liste des scores.

R. db.getCollection('restaurants').find(
{"borough": "Queens",
"cuisine":"Greek",
"grades": $all:[{"$elemMatch" :{"grade": 'C', "score": { $lt: 35 }}}]}},
{"name": 1, "borough": 1, "grades": 1, "_id": 0})

Q. Obtenir le nom et quartier des restaurants ayant eu un C à leur dernière évaluation (elle est en première position dans la liste). La liste résultat sera triée par nom des restaurants

R. db.getCollection('restaurants').find(
{"grades.0.grade": 'C'},
{"name": 1, "borough": 1, "grades": 1, "_id": 0}
).sort({"name": 1})

Q. Obtenir le nombre de restaurant par quartier ayant eu un C à leur dernière évaluation. Vous afficherez le résultat par ordre alphabétiques des quartiers.

R. varMatch = {$match: {"grades.0.grade": 'C'}}
varGroup = {$group: {"_id":"$borough", "res":{"$sum": 1}}}
varSort = {$sort: {"_id": 1}}
db.getCollection('restaurants').aggregate([varMatch, varGroup, varSort])

Q. Donner le score moyen des restaurants par quartier. Vous afficherez le résultat par ordre décroissant.

R. varUnwind = {$unwind:"$grades"}
varGroup = {$group: {"_id":"$borough", "mean":{$avg: "$grades.score"}}}
varSort = {$sort: {"_id": -1}}
db.getCollection('restaurants').aggregate([varUnwind, varGroup, varSort])

3 Jointure

Q. Tester l'opération lookup permettant de faire des jointures entre des collections. Situer-la par rapport aux différents types de jointure de SQL. Vous développerez un jeu de test en utilisant des produits et des commandes pour justifiez vos réponses.

R. Non réalisé

4 MapReduce

Q. Donner le script permettant de calculer la moyenne des notes obtenues lors des évaluations A par type de cuisine.

R.
```
var mapFunction = function (){
    for(let i = 0; i < this.grades.length; i++){
      if(this.grades[i].grade == 'A'){
        emit(this.cuisine, this.grades[i].score);
      }
    }
};
var reduceFunction = function (key, values) {
  return Array.avg(values);
};
var queryParam = {query : {}, out : "result_set"}
db.restaurants.mapReduce(mapFunction, reduceFunction, queryParam);
db.result_set.find();
```

Q. Donner le script permettant de donner le nombre d'évaluation A par année et par quartier

R.
```
var mapFunction = function (){
    for(let i = 0; i < this.grades.length; i++){
      if(this.grades[i].grade == 'A'){
        var id = {year: this.grades[i].date.getFullYear(), borough: this.borough};
        emit(id, 1);
      }
    }
};
var reduceFunction = function (key, values) {
  return Array.sum(values);
};
var queryParam = {query : {}, out : "result_set"}
db.restaurants.mapReduce(mapFunction, reduceFunction, queryParam);
db.result_set.find();
```