# Introduction

Soit salles , la collection construite de la manière suivante :

db.salles.insertMany([

   {

       "\_id": 1,

       "nom": "AJMI Jazz Club",

       "adresse": {

           "numero": 4,

           "voie": "Rue des Escaliers Sainte-Anne",

           "codePostal": "84000",

           "ville": "Avignon",

           "localisation": {

               "type": "Point",

               "coordinates": [43.951616, 4.808657]

           }

       },

       "styles": ["jazz", "soul", "funk", "blues"],

       "avis": [{

               "date": new Date('2019-11-01'),

               "note": NumberInt(8)

           },

           {

               "date": new Date('2019-11-30'),

               "note": NumberInt(9)

           }

       ],

       "capacite": NumberInt(300),

       "smac": true

   }, {

       "\_id": 2,

       "nom": "Paloma",

       "adresse": {

           "numero": 250,

           "voie": "Chemin de l'Aérodrome",

           "codePostal": "30000",

           "ville": "Nîmes",

           "localisation": {

               "type": "Point",

               "coordinates": [43.856430, 4.405415]

           }

       },

       "avis": [{

               "date": new Date('2019-07-06'),

               "note": NumberInt(10)

           }

       ],

       "capacite": NumberInt(4000),

       "smac": true

   },

    {

       "\_id": 3,

       "nom": "Sonograf",

       "adresse": {

           "voie": "D901",

           "codePostal": "84250",

           "ville": "Le Thor",

           "localisation": {

               "type": "Point",

               "coordinates": [43.923005, 5.020077]

           }

       },

       "capacite": NumberInt(200),

       "styles": ["blues", "rock"]

   }

])

# Suppressions, insertions et mises à jour de documents

*Exercice 1*

Affichez l’identifiant et le nom des salles qui sont des SMAC.

*Exercice 2*

Affichez le nom des salles qui possèdent une capacité d’accueil strictement supérieure à 1000 places.

*Exercice 3*

Affichez l’identifiant des salles pour lesquelles le champ adresse ne comporte pas de numéro.

*Exercice 4*

Affichez l’identifiant puis le nom des salles qui ont exactement un avis.

*Exercice 5*

Affichez tous les styles musicaux des salles qui programment notamment du blues.

*Exercice 6*

Affichez tous les styles musicaux des salles qui ont le style « blues » en première position dans leur tableau styles.

*Exercice 7*

Affichez la ville des salles dont le code postal commence par 84 et qui ont une capacité strictement inférieure à 500 places (pensez à utiliser une expression régulière).

*Exercice 8*

Affichez l’identifiant pour les salles dont l’identifiant est pair ou le champ avis est absent.

*Exercice 9*

Affichez le nom des salles dont au moins un des avis comporte une note comprise entre 8 et 10 (tous deux inclus).

*Exercice 10*

Affichez le nom des salles dont au moins un des avis comporte une date postérieure au 15/11/2019 (pensez à utiliser le type JavaScript Date).

*Exercice 11*

Affichez le nom ainsi que la capacité des salles dont le produit de la valeur de l’identifiant par 100 est strictement supérieur à la capacité.

*Exercice 12*

Affichez le nom des salles de type SMAC programmant plus de deux styles de musiques différents en utilisant l’opérateur $where qui permet de faire usage de JavaScript.

*Exercice 13*

Affichez les différents codes postaux présents dans les documents de la collection salles.

*Exercice 14*

Mettez à jour tous les documents de la collection salles en rajoutant 100 personnes à leur capacité actuelle.

*Exercice 15*

Ajoutez le style « jazz » à toutes les salles qui n’en programment pas.

*Exercice 16*

Retirez le style «funk» à toutes les salles dont l’identifiant n’est égal ni à 2, ni à 3.

*Exercice 17*

Ajoutez un tableau composé des styles «techno» et « reggae » à la salle dont l’identifiant est 3.

*Exercice 18*

Pour les salles dont le nom commence par la lettre P (majuscule ou minuscule), augmentez la capacité de 150 places et rajoutez un champ de type tableau nommé contact dans lequel se trouvera un document comportant un champ nommé telephone dont la valeur sera « 04 11 94 00 10 ».

*Exercice 19*

Pour les salles dont le nom commence par une voyelle (peu importe la casse, là aussi), rajoutez dans le tableau avis un document composé du champ date valant la date courante et du champ note valant 10 (double ou entier). L’expression régulière pour chercher une chaîne de caractères débutant par une voyelle suivie de n’importe quoi d’autre est [^aeiou]+$.

*Exercice 20*

En mode *upsert*, vous mettrez à jour tous les documents dont le nom commence par un z ou un Z en leur affectant comme nom « Pub Z », comme valeur du champ capacite 50 personnes (type entier et non décimal) et en positionnant le champ booléen smac à la valeur « false ».

*Exercice 21*

Affichez le décompte des documents pour lesquels le champ \_id est de type « objectId ».

*Exercice 22*

Pour les documents dont le champ \_id n’est pas de type « objectId », affichez le nom de la salle ayant la plus grande capacité. Pour y parvenir, vous effectuerez un tri dans l’ordre qui convient tout en limitant le nombre de documents affichés pour ne retourner que celui qui comporte la capacité maximale.

*Exercice 23*

Remplacez, sur la base de la valeur de son champ \_id, le document créé à l’exercice 20 par un document contenant seulement le nom préexistant et la capacité, que vous monterez à 60 personnes.

*Exercice 24*

Effectuez la suppression d’un seul document avec les critères suivants : le champ \_id est de type « objectId » et la capacité de la salle est inférieure ou égale à 60 personnes.

*Exercice 25*

À l’aide de la méthode permettant de trouver un seul document et de le mettre à jour en même temps, réduisez de 15 personnes la capacité de la salle située à Nîmes.

Correction

# Suppressions, insertions et mises à jour de documents

*Exercice 1*

db.salles.find({"smac": true}, {"nom": 1})

*Exercice 2*

db.salles.find({"capacite": {$gt: 1000}}, {"\_id": 0, "nom": 1})

*Exercice 3*

db.salles.find({"adresse.numero": {$exists: false}}, {"\_id": 1})

*Exercice 4*

db.salles.find({"avis": {$size: 1}}, {"\_id": 1, "nom": 1})

*ou*

db.salles.find({"avis": {$size: 1}}, {"nom": 1})

*Exercice 5*

db.salles.find({"styles": "blues"}, {"\_id": 0, "styles": 1})

*Exercice 6*

db.salles.find({"styles.0": "blues"}, {"\_id": 0, "styles": 1})

*Exercice 7*

db.salles.find({

   $and: [{

       "adresse.codePostal": /^84/,

       "capacite": {$lt: 500}

   }]

}, {

   "\_id": 0,

   "adresse.ville": 1

})

*ou, plus simplement :*

db.salles.find({

   "adresse.codePostal": /^84/,

   "capacite": {$lt: 500}

}, {

   "\_id": 0,

   "adresse.ville": 1

})

*Exercice 8*

db.salles.find({

   $or: [{

       "\_id": {

           $mod: [2, 0]

       }

   }, {

       "avis": {

           $exists: false

       }

   }]

}, {

       "\_id": 1

})

*Exercice 9*

db.salles.find({

   "avis": {

       $elemMatch: {

           "note": {

               $gte: 8,

               $lte: 10

           }

       }

   }

}, {

   "\_id": 0,

   "nom": 1

})

*Exercice 10*

db.salles.find({"avis.date": {

   $gt: new Date('2019-11-15')}

}, {

   "\_id": 0,

   "nom": 1

})

*Ou bien, sur le modèle de l’exercice précédent :*

db.salles.find({

   "avis": {

       $elemMatch: {

           "date": { $gt: new Date('2019-11-15')}

       }

   }

}, {

   "\_id": 0,

   "nom": 1

})

*Exercice 11*

db.salles.find({

   $expr: { $gt: [ {$multiply: ["$\_id", 100]}, "$capacite"]}

}, {

   "\_id": 0,

   "nom": 1,

   "capacite": 1

})

*Exercice 12*

db.salles.find({

   $and: [{

       "styles": {

           $exists: true

       }

   }, {

       "smac": true

   }]

}, {

   $where: 'this.styles.length > 2',

   "\_id": 0,

   "nom": 1

})

*Exercice 13*

db.salles.distinct("adresse.codePostal")

*Exercice 14*

db.salles.updateMany({}, {$inc: {"capacite": NumberInt(100)}})

Attention, si vous omettez de préciser le type NumberInt, votre champ capacite sera transtypé en double.

*Exercice 15*

db.salles.updateMany({"styles": {$ne: "jazz"}}, {$push:

{"styles": "jazz"}})

*Exercice 16*

db.salles.updateMany({"\_id": {$nin: [2,3]}}, {$pull:

{"styles": "funk"}})

*Exercice 17*

db.salles.updateOne(

   {"\_id": 3},

   {$addToSet: {"styles": {$each: [ "techno", "reggae" ]}}}

)

*Exercice 18*

db.salles.updateMany(

   {"nom": /^p/i},

   {

     $inc: {"capacite" : NumberInt(150)},

     $set: { "contact.telephone" : "04 11 94 00 10" }

   }

)

*Exercice 19*

db.salles.updateMany(

   {"nom": /[^aeiou]+$/i},

   { $push: { "avis" : {"date": new Date(), "note": 10} } }

)

Si vous souhaitez que la note soit de type entier, utilisez NumberInt.

*Exercice 20*

db.salles.updateOne(

   {"nom": /^z/i},

   { $set: { "nom" : "Pub Z", "smac": false, "capacite": NumberInt(50) } },

   { "upsert": true}

)

*Exercice 21*

db.salles.find({"\_id": {$type: "objectId"}}).count()

*Exercice 22*

db.salles.find({

   "\_id": {

       $not: {

           $type: "objectId"

       }

   }

}, {

   "\_id": 0,

   "nom": 1

}).sort({"capacite": -1}).limit(1)

*Exercice 23*

db.salles.replaceOne(

   {"\_id": ObjectId("5d32f67d81d43db61fc73696")},

   {"nom" : "Pub Z", "capacite": NumberInt(60)}

)

*Exercice 24*

db.salles.remove({"\_id": {$type: "objectId"}, "capacite": {$lte: 60}}, true)

*Exercice 25*

db.salles.findOneAndUpdate(

  { "adresse.ville" : "Nîmes" },

  { $inc: { "capacite" : NumberInt(-15) } }

)

# Validation des documents

*Exercice 1*

Modifiez la collection salle afin que soient dorénavant validés les documents destinés à y être insérés ; cette validation aura lieu en mode « strict » et portera sur les champs suivants :

* nom sera obligatoire et devra être de type chaîne de caractères.
* capacite sera obligatoire et devra être de type entier (int).
* Dans le champ adresse, les champs codePostal et ville, tous deux de type chaîne de caractères, seront obligatoires.

Que constatez-vous lors de la tentative d’insertion suivante, et quelle en est la cause ?

db.salles.insertOne(

{"nom": "Super salle", "capacite": 1500, "adresse": {"ville": "Musiqueville"}}

)

Que proposez-vous pour régulariser la situation ?

*Exercice 2*

Rajoutez à vos critères de validation existants un critère supplémentaire : le champ \_id devra dorénavant être de type entier (int) ou ObjectId.

Que se passe-t-il si vous tentez de mettre à jour l’ensemble des documents existants dans la collection à l’aide de la requête suivante :

db.salles.updateMany({}, {$set: {"verifie": true}})

Supprimez les critères rajoutés à l’aide de la méthode delete en JavaScript.

*Exercice 3*

Rajoutez aux critères de validation existants le critère suivant :

Le champ smac doit être présent OU les styles musicaux doivent figurer parmi les suivants : "jazz", "soul", "funk" et "blues".

Que se passe-t-il lorsque nous exécutons la mise à jour suivante ?

db.salles.update({"\_id": 3}, {$set: {"verifie": false}})

Correction :

# Validation des documents

*Exercice 1*

Voilà notre commande de création d’une validation sur salles :

db.runCommand( {

  collMod: "salles",

  validator: { $jsonSchema: {

     bsonType: "object",

     required: [ "nom", "capacite", "adresse.codePostal", "adresse.ville"],

     properties: {

        "nom": {

           bsonType: "string",

           description: "Chaîne de caractères - requis"

        },

        "capacite": {

           bsonType: "int",

           description: "Entier - requis"

        },

        "adresse.codePostal": {

           bsonType: "string",

           description: "Chaîne de caractères - requis"

        },

       "adresse.ville": {

           bsonType: "string",

           description: "Chaîne de caractères - requis"

        },

     }

  } }

})

Une autre version, nous permettant de signifier que codePostal et ville doivent impérativement exister, mais cette fois-ci au sein du champ adresse et plus du document racine, serait la suivante :

db.runCommand( {

  collMod: "salles",

  validator: { $jsonSchema: {

     bsonType: "object",

     required: [ "nom", "capacite"],

     properties: {

        "nom": {

           bsonType: "string",

           description: "Chaîne de caractères - requis"

        },

        "capacite": {

           bsonType: "int",

           description: "Entier - requis"

        },

       "adresse": {

          bsonType: "object",

          required: [ "codePostal", "ville" ],

          properties: {

           "codePostal": {

               bsonType: "string",

               description: "Chaîne de caractères - requis"

           },

           "ville": {

               bsonType: "string",

               description: "Chaîne de caractères - requis"

           },

          }

       }

     }

  } }

})

Dans un cas comme dans l’autre, lorsque nous tentons d’insérer le document suivant :

{"nom": "Super salle", "capacite": 1500, "adresse": {"ville":

"Musiqueville"}}

L’erreur suivante s’affiche :

WriteError({

       "index" : 0,

       "code" : 121,

       "errmsg" : "Document failed validation",

       "op" : {

               "\_id" : ObjectId("5d343d2494667ef0984b4810"),

               "nom" : "Super salle",

               "capacite" : 1500,

               "adresse" : {

                       "ville" : "Musiqueville",

                       "codePostal" : "84310"

               }

       }

})

En effet, nous avons requis un type entier pour le champ capacite alors que nous insérons un décimal. En effet, nous savons que MongoDB traite par défaut les numériques comme des nombres décimaux. Pour mettre les documents résidant au sein de notre collection en conformité avec cette nouvelle validation, nous n’avons pas d’autre choix que de procéder à des mises à jour unitaires. Ce n’est pas très gênant à ce stade, car leur nombre est très petit. Vous imaginez sans peine qu’il en ira tout autrement sur des collections comportant plusieurs centaines de milliers (ou millions) de documents !

db.salles.updateOne({"\_id":1}, {$set: {"capacite": NumberInt(300)}})

db.salles.updateOne({"\_id":2}, {$set: {"capacite": NumberInt(3985)}})

db.salles.updateOne({"\_id":3}, {$set: {"capacite": NumberInt(200)}})

db.salles.updateOne(

   {"\_id": ObjectId("5d343d4694667ef0984b4811")},

   {$set: {"capacite": NumberInt(50)}}

)

*Exercice 2*

Le code JavaScript permettant la mise à jour des critères de validation est quasiment le même que celui vu dans le chapitre consacré à la validation des documents, à ceci près que nous spécifions ici une énumération de types au sein d’un tableau :

let validateur = db.getCollectionInfos({name: "salles"})[0].options.validator;

validateur.$jsonSchema.properties.\_id = {

   "bsonType": ["int", "objectId"],

   "description": "Entier ou ObjectId - requis"

}

db.runCommand({

 "collMod": "salles",

 "validator": validateur

})

Lorsque nous exécutons la requête de mise à jour de l’ensemble des documents de la collection, l’erreur de validation suivante est levée :

WriteError({

       "index" : 0,

       "code" : 121,

       "errmsg" : "Document failed validation",

       "op" : {

               "q" : {

               },

               "u" : {

                       "$set" : {

                               "verifie" : true

                       }

               },

               "multi" : true,

               "upsert" : false

       }

})

La raison est simple : les documents qui existaient avant la mise en place de la validation n’ont pas été passés au crible de celle-ci, ils ne le sont qu’une fois qu’ils sont altérés, en l’occurrence à travers une mise à jour. Comme les identifiants présents dans la collection sont tous de type décimal (le type par défaut donné aux numériques sous MongoDB) et non entier ou ObjectId comme exigé, la validation échoue logiquement !

Le problème posé est de taille : vous le savez maintenant, le champ \_id est un champ spécial et immuable, ce qui signifie donc en substance que vous ne pourrez pas le modifier. Si vous tentez une mise à jour du champ identifiant pour le transtyper en entier ou en ObjectId, vous obtiendrez un message d’erreur suffisamment explicite :

db.salles.updateOne(

   {"\_id": 1},

   {$set: {"\_id" : ObjectId("5cb9c7fc5c5308e1d6dcb461")}}

)

WriteError({

       "index" : 0,

       "code" : 66,

**"errmsg" : "Performing an update on the path '\_id' would modify the**

**immutable field '\_id'",**

       "op" : {

               "q" : {

                       "\_id" : 1

               },

               "u" : {

                       "$set" : {

                               "\_id" : ObjectId("5cb9c7fc5c5308e1d6dcb461")

                       }

               },

               "multi" : false,

               "upsert" : false

       }

})

Pour supprimer notre clause de validation sur le champ \_id, il nous faudra écrire :

delete validateur.$jsonSchema.properties.\_id

Puis rattacher notre validateur ainsi modifié à la collection :

db.runCommand({

 "collMod": "salles",

 "validator": validateur

})

*Exercice 3*

let validateur = db.getCollectionInfos({name: "salles"})[0].options.validator

validateur.$or = [

   { "smac": { $exists: true } },

   { "styles": { $in: ["jazz", "soul", "funk", "blues"] } }

]

db.runCommand({

 "collMod": "salles",

 "validator": validateur

})

La requête de mise à jour échouera au double motif que le document ne comporte pas le champ smac et qu’il ne possède que des styles musicaux qui ne sont pas dans la liste des styles que nous autorisons.

# Les index

*Exercice 1*

Un bref examen de vos fichiers journaux a révélé que la plupart des requêtes effectuées sur la collection salles cible des capacités ainsi que des départements, comme ceci :

db.salles.find({"capacite": {$gt: 500}, "adresse.codePostal": /^30/})

db.salles.find({"adresse.codePostal": /^30/, "capacite": {$lte: 400}})

Que proposez-vous comme index qui puisse couvrir ces requêtes ?

Détruisez ensuite l’index créé.

*Exercice 2*

Nous souhaitons pouvoir chercher dans les adresses des noms de voie, de cette façon :

db.salles.find({$text: {$search: "chemin"}})

Quel type d’index préconisez-vous ?

*Exercice 3*

Vous souhaitez pouvoir exécuter ce genre de requête retournant les documents dans l’ordre croissant d’éloignement par rapport à un point donné, quel type d’index allez-vous devoir créer ?

db.salles.find(

   {

       "localisation": {

           $nearSphere: {

               "type" : "Point",

               "coordinates" : [ 43.923005, 5.020077 ]

           },

           $minDistance: 0.5/6378,

           $maxDistance: 1000/6378

       }

   }, {"\_id": 0, "nom": 1}

)

*Exercice 4*

Les requêtes que vous recevez ciblent pour la plupart le nom et le département des salles dont la capacité est strictement supérieure à 100 personnes, comme dans cet exemple :

db.salles.find({"nom": /^L/, "capacite": 150})

db.salles.find({"nom": /^p/, "adresse.codePostal": /^30/,

"capacite": 200})

Créez un index partiel nommé idx\_part\_nom\_ville\_capacite qui puisse être utilisé par les requêtes précédentes.

*Exercice 5*

Vous avez décidé que votre collection salles ne comportera dorénavant qu’un seul document par ville, écrivez l’index qui matérialisera cette contrainte.

Correction :

# Les index

*Exercice 1*

L’index suivant assurera une couverture de nos requêtes les plus fréquentes :

db.salles.createIndex({ "capacite": 1, "adresse.codePostal": 1 })

Un explain appliqué à nos requêtes montre que le collscan qui était fait avant la mise en place de l’index a disparu au profit de notre index dont le nom par défaut est capacite\_1\_adresse.codePostal\_1.

Cet index pourra également couvrir les tris suivants :

db.salles.find({}).sort({"capacite": 1})

db.salles.find({}).sort({ "capacite": 1, "adresse.codePostal": 1 })

Pour détruire notre index, il suffira d’exécuter :

db.salles.dropIndex("capacite\_1\_adresse.codePostal\_1")

*Exercice 2*

Le doute n’est pas permis, la solution consiste à créer un index textuel ! Nous prendrons soin de dire que la langue utilisée dans notre collection est le français :

db.salles.createIndex({"adresse.voie": "text"}, {"default\_language": "french"})

*Exercice 3*

Il va nous falloir créer un un index de type 2dsphere pour prendre en charge cette requête géospatiale :

db.salles.createIndex({"localisation": "2dsphere"})

Si nous ne le faisons pas, la requête produira une erreur :

 planner returned error: unable to find index for $geoNear query

*Exercice 4*

Notre index partiel va consister à cibler un sous-ensemble des valeurs prises par le champ capacite, en l’occurrence celles supérieures à 100 personnes. Comme le champ nom figure dans les deux requêtes, il est un candidat parfait pour constituer le préfixe de notre index idx\_part\_nom\_ville\_capacite :

db.salles.createIndex(

  {"nom": 1, "adresse.ville": 1},

  {"name": "idx\_part\_nom\_ville\_capacite",

   "partialFilterExpression": {"capacite": {$gt: 100}}

  }

)

*Exercice 5*

Il s’agit bien évidemment d’un index unique ! La contrainte portera sur le champ ville du document adresse :

db.salles.createIndex(

   { "adresse.ville": 1 },

   { "name": "id\_uniq\_adresse\_ville", "unique": true }

)