1. Afficher le nom des étudiants habitant à ...

db.etudiants.find( {‘adresse.ville’ : ‘chambery},{ ‘nom’ : 1})

1. Afficher les étudiants habitant à ... (sans l’ID)

db.etudiants.find( {‘adresse.ville’ : ‘chambery’},{ ‘\_id’ : 0})

1. Afficher les étudiants habitant à ...ou à ...

db.etudiants.find( {$or :[{‘adresse.ville’ : ‘chambery’},{‘adresse.ville’ : ‘Lyon’}]})

1. Afficher les étudiants habitant à ... ou à ... ordonné par ... décroissant

db.etudiants.find( {$or :[{‘adresse.ville’ : ‘chambery’},{‘adresse.ville’ : ‘Lyon’}]}).sort({‘nom’ :-1})

1. Afficher les étudiants dont le ... est parmi ... (utiliser $in)

db.etudiants.find( {‘nom’ :{$in :[‘caille , ‘gamain]}})

1. Afficher les étudiants dont le ... est ... et la ville est ...

db.etudiants.find( {$and : [{‘nom’ : ‘gamain},{‘adresse.ville’ : ‘lyon}]})

1. Afficher tous les étudiants n’habitant pas à ...

db.etudiants.find( {‘adresse.ville’ :{‘$not’:/annecy/}})

1. Afficher les étudiants n’habitant ni à ... , ni à ...

db.etudiants.find( {$and :[{‘adresse.ville’ :{‘$not’:/Dijon/}}:{‘adresse.ville’ :{‘$not’:/Annecy/}}]})

1. Afficher les étudiants dont l'un des prénoms n'est ni ... ni ...

db.etudiants.find({‘prenoms’ :{$not’ :{$in :[/antoine/,/loic/]}}})

1. Afficher les étudiants qui habitent à ... mais pas dans la rue ...

db.etudiants.find({$and :[{‘adresse.ville’ : ‘chambéry},{adresse.rue’ :{‘$not’:/croix d or/}}]})

1. Afficher les étudiants ayant plus de ... ans d’expérience

db.etudiants.find({‘annee experience’ : {$gt : 5}})

1. Afficher les étudiants dont ... est renseigné

db.etudiants.find({‘annee experience’ : {$exists : true }})

1. Afficher les étudiants dont l’un des prénoms commence par ...

db.etudiants.find({‘prenoms’ :{$regex:/^ant/}})

1. Afficher les étudiants à partir du numéro 2 (utiliser $skip)

db.etudiants.aggregate([{$skip : 1 }])

1. Trier les étudiants par ... décroissant

db.etudiants.aggregate([{$sort : -1 }])

1. Afficher le nombre moyen d’années d’expérience des étudiants

db.etudiants.aggregate([{

"$group" : {

"\_id" : null,

"avg\_ae" : { "$avg" : "$annee experience" }

}

}])

1. Afficher le nombre moyen d’années d’expérience par ...

db.etudiants.aggregate([{

"$group" : {

"\_id" : "$nom",

"avg\_ae" : { "$avg" : "$annee experience" }

}

}])

1. Afficher le nombre d’étudiants habitant à ...

db.etudiants.aggregate([

{$group : { \_id : "$adresse.ville" , myCount : {$sum:1}}},

{$project : {\_id: 1 }}

])

1. Afficher le nombre d’étudiants dont le champ ... est renseigné

db.etudiants.find({"annee experience" : {$exists:true} })

1. Afficher le nombre d’étudiants ayant un nom différent

db.etudiants.distinct(

"nom"

).length

1. Afficher par ordre décroissant, pour chaque prénom existant, le nombre d'étudiants qui portent ce prénom (utiliser $unwind)

db.etudiants.aggregate([

{$group : { \_id : "$nom" , myCount : {$sum:1}}},

{$sort : { myCount: -1}},

{$project : {\_id: 1 , myCount:1}}

])

**Créer les formations :**

var formation

= {// Création d’un objet JSON

"\_id" :"lpbdd",

"intitule" : "Licence professionnelle Bases de données",

"Annee\_creation" : "2011",

}

db.formation.insert(formation)

var formation

= {// Création d’un objet JSON

"\_id" :"dutinfo",

"intitule" : " DUT Informatique",

"Annee\_creation" : "2005",

"Composante" : "iut annecy"

}

db.formation.insert(formation)

var formation

= {// Création d’un objet JSON

"\_id" :"masterstic",

"intitule" : "Master Informatique et Systèmes coopératifs",

"Annee\_creation" : "2005",

"Composante" : "SCEM"

}

db.formation.insert(formation)

var formation

= {// Création d’un objet JSON

"\_id" :"irc",

"intitule" : "Informatique et réseaux de communication",

"Annee\_creation" : "2005",

}

db.formation.insert(formation)

**Ajout des formations :**

db.etudiants.update({\_id: { $in: [1, 2] }}, { $set: { formation: 'dutinfo' } }, { multi: true })

db.etudiants.update({ \_id:{ $nin: [1, 2] } },{ $set: { formation: 'lpbdd' } }, { multi: true })

**Jointure:**

db.etudiants.aggregate([

{

$lookup :

{

from : "formation",

localField : "formation",

foreignField : "\_id",

as : "nom\_formation"

}

}

])

**Indexes :**

**Index primaire** : Index sur l’id (\_id)

automatique à la création de la « table »

**Index simple** : Index sur un champ

db.etudiants.createIndex({"nom" :1})

**Index composé** : Index sur un ensemble de champ

db.etudiants.createIndex({"nom" :1,”prenom” : 1})

**Index MutltiKey**: Index sur un tableau

db.etudiants.createIndex({"adresse" :1})

**Index texte** : Indexe sur les chaines de cratères

**Index géo-spatiaux** : Index sur les éléments géo spatiaux comme les 2dsphere ou 3dsphere

**Index hachés** : Index sur les éléments hachés

**Options :**

**Time To Live** : Spécifie une valeur pour contrôler combien de temps MongoDB retient des documents dans la collection

**Unique** : Empêche d’avoir deux fois la même valeur sur le champ indexé

**Partiel** : Référence uniquement les éléments qui valident un filtre

**Epars** : Référence uniquement les éléments avec un champ spécifié

**Insensibilité à la case** : Permet de faire des recherches sur l’index sans sensibilité à la case

db.etudiants.createIndex( {« nom » : 1},

{ collation : {

strength : 2

}

}

)

**setting.py**

# Let's just use the local mongod instance. Edit as needed.

# Please note that MONGO\_HOST and MONGO\_PORT could very well be left

# out as they already default to a bare bones local 'mongod' instance.

MONGO\_HOST = 'localhost'

MONGO\_PORT = 27017

# Skip this block if your db has no auth. But it really should.

#MONGO\_USERNAME = '<your username>'

#MONGO\_PASSWORD = '<your password>'

# Name of the database on which the user can be authenticated,

# needed if --auth mode is enabled.

#MONGO\_AUTH\_SOURCE = 'etudiants'

MONGO\_DBNAME = 'bdetudiant'

# Enable reads (GET), inserts (POST) and DELETE for resources/collections

# (if you omit this line, the API will default to ['GET'] and provide

# read-only access to the endpoint).

RESOURCE\_METHODS = ['GET', 'POST', 'DELETE']

# Enable reads (GET), edits (PATCH), replacements (PUT) and deletes of

# individual items (defaults to read-only item access).

ITEM\_METHODS = ['GET', 'PATCH', 'PUT', 'DELETE']

schema = {

# Schema definition, based on Cerberus grammar. Check the Cerberus project

# (https://github.com/pyeve/cerberus) for details.

'nom': {

'type': 'string'

},

# 'prenoms' is a list, and can only contain values from 'allowed'.

'prenoms': {

'type': 'list',

'allowed': ["author", "contributor", "copy"],

},

# An embedded 'strongly-typed' dictionary.

'adresse': {

'type': 'dict',

'schema': {

'numero': {'type': 'integer'},

'rue': {'type': 'string'},

'ville': {'type': 'string'},

'cp': {'type': 'integer'}

},

},

'date naissance': {

'type': 'datetime',

},

}

etudiants = {

# 'title' tag used in item links. Defaults to the resource title minus

# the final, plural 's' (works fine in most cases but not for 'people')

'item\_title': 'etudiants',

# by default the standard item entry point is defined as

# '/people/<ObjectId>'. We leave it untouched, and we also enable an

# additional read-only entry point. This way consumers can also perform

# GET requests at '/people/<lastname>'.

'additional\_lookup': {

'url': 'regex("[\w]+")',

'field': 'nom'

},

# We choose to override global cache-control directives for this resource.

'cache\_control': 'max-age=10,must-revalidate',

'cache\_expires': 10,

# most global settings can be overridden at resource level

'resource\_methods': ['GET', 'POST'],

'schema': schema

}

DOMAIN = {'etudiants': etudiants}