MongoDB
Sommaire

E.Porcq : R5;10 MongoDB
Département : IUT Caen Informatique
Année universitaire : 2025-2026

Sources bibliographiques :

- https://fr.wikipedia.org/wiki/MongoDB
- https://openclassrooms.com/fr/courses/4462426-maitrisez-les-bases-de-donnees-nosql/4474601-decouvrez-le-fonctionnement-de-mongodb

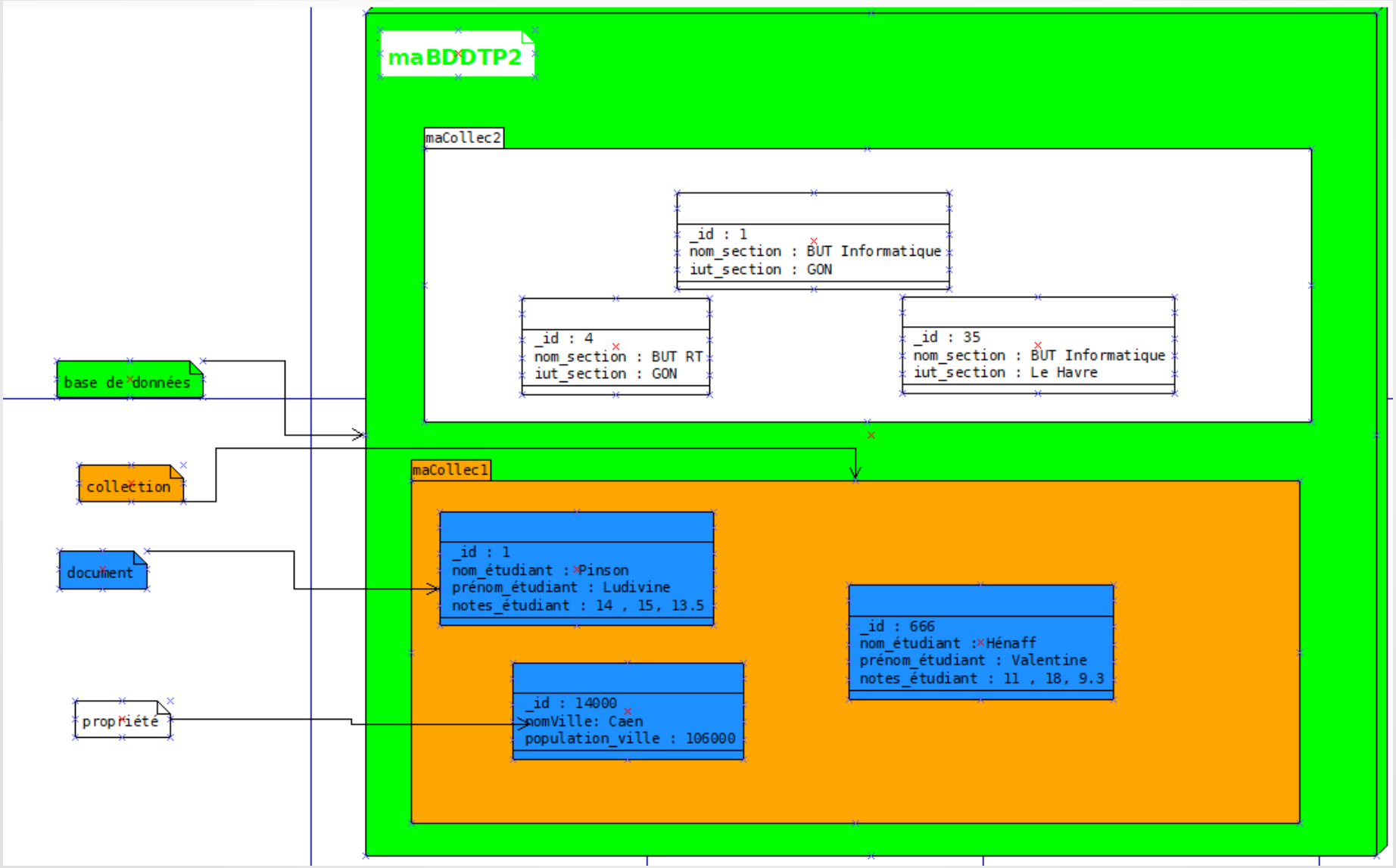
1) Présentation : MongoDB ...

- doit son nom de l'anglais humongous (énorme)
- a été développé en 2007 (1ère version 2009)
- est écrit en C++
- est un système de gestion de base de données NoSQL
- est orienté documents
- ne nécessite pas de schéma prédéfini des données.
- Est distribué sous licence propriétaire SSPL pour le serveur et en licence libre Apache pour les pilotes.
- Est le premier SGBD NoSQL du marché
- permet de manipuler des objets structurés au format BSON (JSON binaire)
- Est en version stable avec le code 7.0.12 (la version installée au campus 3 est la 6.0.8)

2) Le modèle orienté document

- Les données sont enregistrées dans des documents. On peut considérer par analogie que les documents sont équivalentes aux enregistrements dans des tables
- Les documents sont enregistrés dans des collections que l'on peut considérer comme des tables.
- Contrairement aux bases de données relationnelles, les propriétés d'un enregistrement sont libres et peuvent être différentes d'un enregistrement à un autre au sein d'une même collection.
- Le seul champ commun et obligatoire est le champ de clé principale ("id").
- MongoDB ne permet ni les requêtes très complexes standardisées, ni les JOIN, mais permet de programmer des requêtes spécifiques en JavaScript.
- De même, un champ peut contenir un tableau
- Nous sommes donc loin des formes normales en vigueur dans les SGBDR. C'est ce qu'on appelle la dénormalisation.
- La structure d'un document est très simple et se compose de paires clef/valeur
- la clef est le nom du champ, la valeur son contenu (cf format Json)

2) Le modèle orienté document



3) Les documents

- Voici la forme d'un documeent

```
{
    _id: ObjectId("64d3a92fa708a730b7c5afd1"),
    nom: 'Supormoi',
    prenom: 'Steven',
    age: 59
}
```

- On peut imbriquer des champs structurés comme on veut

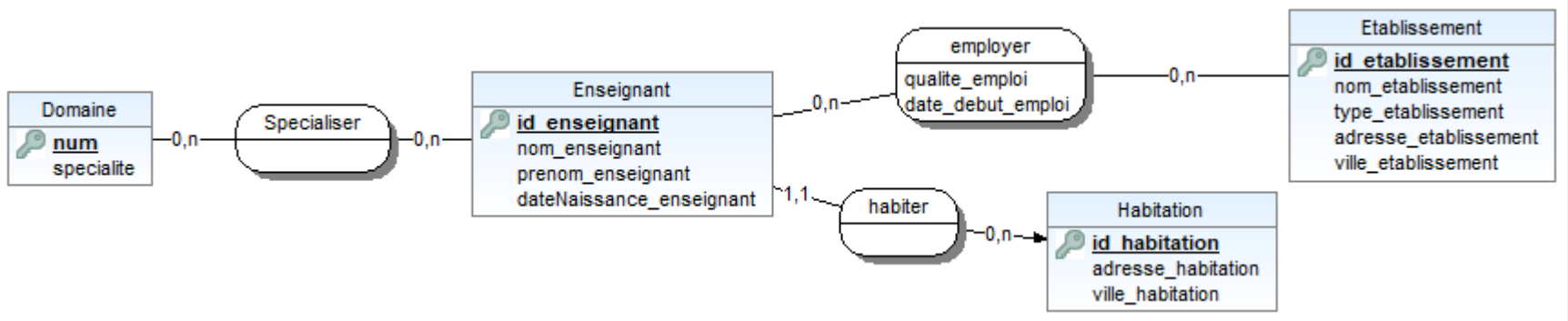
```
{
    _id: "28",
    nom: 'Muda',
    prenom: 'Robert',
    adresse: { rue: '5 rue de Bras', ville: 'Ifs' },
    age: 36
}
```

- Certains champs peuvent contenir des tableaux

```
{
    _id: ObjectId("64d3b11da708a730b7c5afd3"),
    nom: 'Delalune',
    prenom: 'Claire',
    telephone: [ '02 31 23 66 89', '06 30 20 10 15' ]
}
```

4) Relationnel vers JSON : La dénormalisation

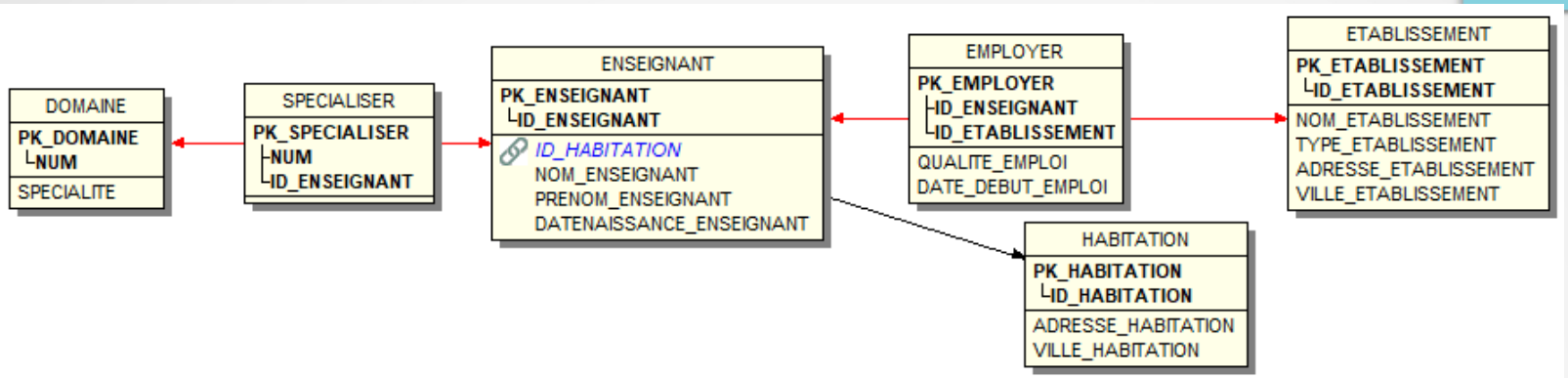
- Pour pallier l'impossibilité de faire des jointures, on va agréger au minimum les associations de type CIF



- Ainsi l'entité Habitation sera intégrée à Enseignant
- Les enseignants sont spécialisés dans 1 ou plusieurs domaines. Ces derniers étant assez peu nombreux, il n'est pas préjudiciable de les ajouter à Enseignant sous forme de tableau
- Il peut être assez coûteux d'intégrer Etablissement dans Enseignant ; pas pour les enseignants ayant été employés dans beaucoup d'établissement mais plutôt à cause du grand nombre de propriétés dans Etablissement.
- L'intégration d' "employer " dans Enseignant ne posera pas de problème.
- L'intégration d'Enseignant dans Etablissement serait par contre une grosse erreur.

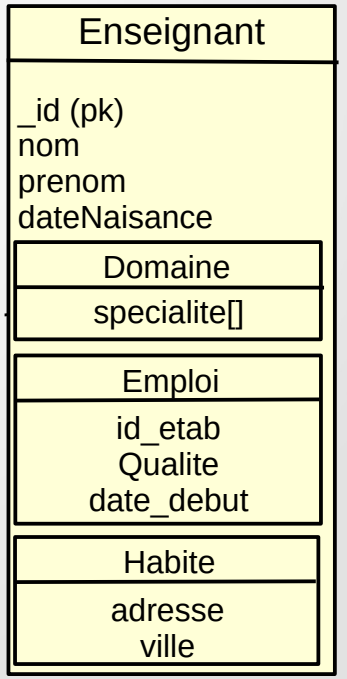
MongoDB
4) Relationnel vers JSON

- Pour un SGBDR, on obtiendrait cela



- Nous ferons une collection spécifique Etablissement
- Un document de la collection Personne aura donc cette forme

```
{
  "_id" : 1,    "nom" : "Houssel",   "prenom" : "Kader",
  "dateNaisance" : "13/12/1990",
  "domaine" : ["SGBD", "NoSQL", "UML"],
  "emploi" : [
    {"id_etablissement" : "35", "qualité" : "Maître de Conférences"
        "date_debut" : "01/09/2012"},
    {"id_etablissement" : "38", "qualité" : "Vacataire",
        "date_debut" : "01/09/2007"}
  ],
  "habite" : {"adresse" : "16 rue de Bras", "ville" : "Ifs"}
}
```



5) le client mongo (ou mongosh depuis mongo 5.0)

- MongoDB est un des rares SGBD NoSQL possédant un mode interactif
- Cet interprète en ligne de commande permet de créer de gérer les collections et manipuler les documents en Javascript ou en Node.js
- Commandes de bases :
  - `maBDDcours> show dbs` * permet de voir les bases de données *

```
admin       40.00 KiB
config      72.00 KiB
local       40.00 KiB
maBDDTP2     8.00 KiB
maBDDTP3     8.00 KiB
maBDDcours  72.00 KiB
```

  - `maBDDcours> use maBDDTP2`

```
switched to db maBDDTP2
```

* la collection sera créée si elle n'existe pas *

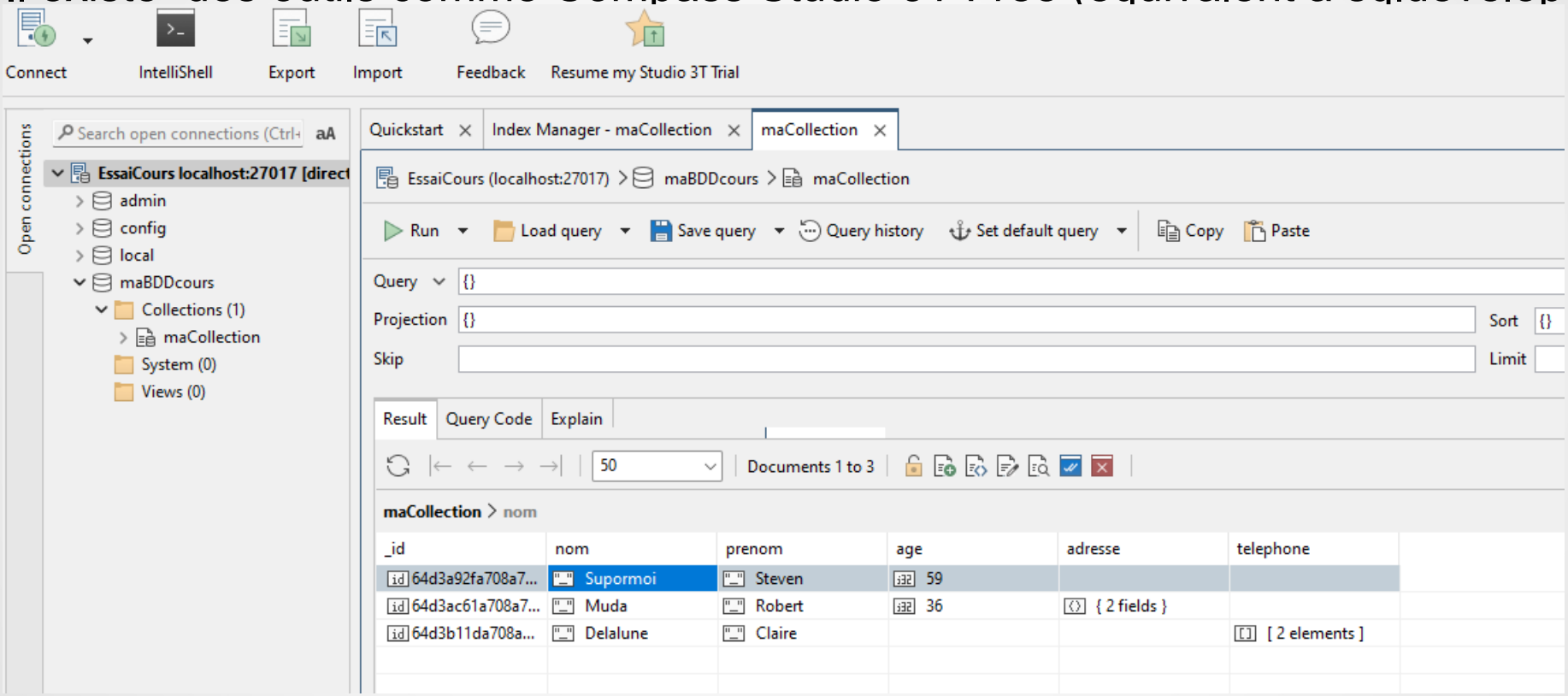
  - `maBDDcours> db.dropDatabase()`
  - `maBDDcours> show collections`

```
essai1
maCollection
```

  - `db.createCollection('nomDeLaCollection')`
  - `db.nomDeLaCollection.drop()`
  - `...`

6) les autres outils

- Il existe des outils comme Compass Studio 3T Free (équivalent à sqldeveloper)



- MongoDB a publié par une annonce la disponibilité générale de Atlas MongoDB lequel est conçu pour travailler avec des clusters multi-cloud pour permettre aux clients de bénéficier facilement du déploiement d'applications multi-cloud.

7) Convertir une table/requête en Json

- Par concaténation, il est possible de fabriquer une collection au format Json
- Il existe des outils pour le réaliser automatiquement
- Avec Oracle, il existe des requêtes pour générer tableaux (json_array) et documents Json (JSON objects)
- Avec une requête, on put créer une vue `vt_classement2023` avec le classement du Tour 2023

```
select * from vt_classement2023;
PLACE       N_COUREUR  N_DOSSARD PRENOM    NOM           EQUIPE                TEMPS
----------------------------------------------------------------------------------
1           2157       1         Jonas     VINGEGAARD    TEAM JUMBO-VISMA      295542
2           2133       11        Tadej     POGACAR       UAE TEAM EMIRATES     295991
3           1233       19        Adam      YATES         UAE TEAM EMIRATES     296198
```

- On exécute la requête de génération Json

```
select json_object( place, n_coureur, n_dossard,prenom, nom, equipe,  TEMPS ) from vt_classement2023;
{"place":1,"n_coureur":2157,"n_dossard":1,"prenom":"Jonas","nom":"VINGEGAARD","equipe":"TEAM JUMBO-VISMA","TEMPS":295542}
{"place":2,"n_coureur":2133,"n_dossard":11,"prenom":"Tadej","nom":"POGACAR","equipe":"UAE TEAM EMIRATES","TEMPS":295991}
{"place":3,"n_coureur":1233,"n_dossard":19,"prenom":"Adam","nom":"YATES","equipe":"UAE TEAM EMIRATES","TEMPS":296198}
```

- On peut enfin l'importer dans MongoDb

```
V:\>mongoimport --db maBDDcours --collection maCollecClassement < classement2023.json
2023-08-11T17:49:58.969+0200    connected to: mongodb://localhost/
2023-08-11T17:49:59.425+0200    150 document(s) imported successfully. 0 document(s) failed to import.
```

8) Projeter des documents d'une collection

- maBDDcours> db.maCollecClassement.findOne()

```
{
  _id: ObjectId("64d665bc25cc56d14c9e6e9c"),
  place: 1,
  n_coureur: 2157,
  n_dossard: 1,
  prenom: 'Jonas',
  nom: 'VINGEGAARD',
  equipe: 'TEAM JUMBO-VISMA',
  TEMPS: 295542
}
```

- maBDDcours> db.maCollecClassement.find()

```
{
  _id: ObjectId("64d665bc25cc56d14c9e6e9c"),
  place: 1,
  n_coureur: 2157,
  n_dossard: 1,
  prenom: 'Jonas',
  nom: 'VINGEGAARD',
  equipe: 'TEAM JUMBO-VISMA',
  TEMPS: 295542
},
{
  _id: ObjectId("64d665bc25cc56d14c9e6e9d"),
  place: 7,
  n_coureur: 2455,
  n_dossard: 71,
  prenom: 'Jai',
  nom: 'HINDLEY',
  equipe: 'BORA-HANSGROHE',
  TEMPS: 296426
},
{
 ...
```

MongoDB

8) Projeter des documents d'une collection

- Il est possible de filtrer les résultats

```
maBDDcours> db.maCollecClassement.find( { "equipe" : "AG2R-CITROEN TEAM" } )
[
  {
    _id: ObjectId("64d665bc25cc56d14c9e6e9e"),
    place: 8,
    n_coureur: 2460,
    n_dossard: 95,
    prenom: 'Felix',
    nom: 'GALL',
    equipe: 'AG2R-CITROEN TEAM',
    TEMPS: 296511
  },
  {
    _id: ObjectId("64d665bc25cc56d14c9e6ea8"),
    place: 17,
    n_coureur: 2177,
    n_dossard: 91,
    ...
```

- Il est possible également de ne projeter que quelques propriétés

```
maBDDcours> db.maCollecClassement.find( { "equipe" : "AG2R-CITROEN TEAM" },{_id:0, nom:1, prenom:1} )
[
  { prenom: 'Felix', nom: 'GALL' },
  { prenom: 'Ben', nom: "O'CONNOR" },
  { prenom: 'Clément', nom: 'BERTHET' },
  { prenom: 'Aurélien', nom: 'PARET-PEINTRE' },
  { prenom: 'Nans', nom: 'PETERS' },
  { prenom: 'Oliver', nom: 'NAESEN' },
  { prenom: 'Stan', nom: 'DEWULF' },
  { prenom: 'Benoît', nom: 'COSNEFROY' }
]
```

MongoDB

8) Projeter des documents d'une collection

- Sans entrer dans les détails, il est possible de réaliser des restrictions complexes

$gt, $gte	>, ≥	Plus grand que (greater than)	"a" : {"$gt" : 10}
$lt, $lte	<, ≤	Plus petit que (less than)	"a" : {"$lt" : 10}
$ne	≠	Différent de (not equal)	"a" : {"$ne" : 10}
$in, $nin	∈, ∉	Fait parti de (ou ne doit pas)	"a" : {"$in" : [10, 12, 15, 18] }
$or	∨	OU logique	"a" : {“$or” : [{"$gt" : 10}, {“$lt” : 5}] }
$and	∧	ET logique	"a" : {“$and” : [{"$lt" : 10}, {“$gt” : 5}] }
$not	¬	Négation	“a" : {“$not” : {"$lt" : 10} }
$exists	∃	La clé existe dans le document	“a” : {“$exists” : 1}
$size		test sur la taille d'une liste (uniquement par égalité)	“a” : {“$size” : 5}

- Exemple

```
maBDDcours> db.maCollecClassement.find( { $and : [ { "place" : { "$lt" : 5 } }  , { "equipe" : { $ne : "UAE TEAM EMIRATES" } } ] } )
    [
      {
        _id: ObjectId("64d665bc25cc56d14c9e6e9c"),
        place: 1,
        n_coureur: 2157,
        n_dossard: 1,
        prenom: 'Jonas',
        nom: 'VINGEGAARD',
        equipe: 'TEAM JUMBO-VISMA',
        TEMPS: 295542
      },
      {
        _id: ObjectId("64d665bc25cc56d14c9e6ea6"),
        place: 4,
        n_coureur: 1098,
        n_dossard: 161,
        prenom: 'Simon',
        nom: 'YATES',
        equipe: 'TEAM JAYCO ALULA',
        TEMPS: 296285
      }
    ]
```

- Il est aussi possible de réaliser des fonctions d'agrégations avec “`aggregate`”

9) Mettre à jour une collection

- Il existe de nombreuses fonctions de mise à jours des données
  - db.<collection>.insertOne()
  - db.<collection>.insertMany()
  - db.<collection>.updateOne()
  - db.<collection>.updateMany()
  - db.<collection>.findAndModify()
  - db.<collection>.replaceOne()
  - db.<collection>.replaceMany()
  - db.<collection>.deleteOne()
  - db.<collection>.deleteMany()
  - db.<collection>.save()