Structure de données, schéma et MongoDB

Institut ICOM
MALIA

2024 – 2025

Juba AGOUN, PhD

Introduction et rappel

- Le modèle relationnel repose sur une stabilité du schéma des données
- Les schémas de données sont toujours amenés à évoluer
- Les bases NoSQL cherchent à assouplir cette contrainte, généralement en proposant une approche dite **sans schéma** , ou à schéma **relaxé.**
- C'est au développeur de décider comment organiser ses données.

Introduction et rappel

- MongoDB se déclare schema-less (sans schéma), le terme peut faire penser qu'un moteur n'a pas de schéma du tout.

Toute donnée complexe manipulée en programmation est une structure, elle a besoin d'un schéma qui décrit cette structure.

- La différence est que, contrairement aux moteurs relationnels où le schéma est explicite (contrôlé par le serveur) et fortement structuré, les moteurs NoSQL possèdent un schéma implicite et semi- structuré

Structure des données

- Avoir une description (formelle) de la structure des données
  - Pouvoir requêter ces données
  - Pouvoir stocker, indexer, etc.

Valider les données (vérifier leur conformité à une description)

Schémas

Contraintes sur les données

- Forme
- Types
- Valeurs

Dans le modèle relationnel

- Nombre et nom des attributs dans une relation
- Type des attributs
- Clés, dépendances

Dans les arbres ?

- Forme = structure : quels type de nœud, à quel endroit
- Types = types de base, mais aussi structure
- Clés : peu / pas utilisé

$$Schéma \leftrightarrow Type$$

JSON Schema

Norme pour (entre autres) spécifier des schémas pour les documents JSON.

- JSON Schema Validation : contraintes de forme et de type
- Syntaxe JSON
- Sémantique des documents JSON Schema (Validation) ≈ types

Types et instances

- Type = description de ce qui est attendu / autorisé
- Sémantique d'un type : ensemble de valeurs correspondant au type
- Instance d'un type : une des valeurs dans la sémantique du type

String

- Type : string
- Sémantique de string : l'ensemble des chaîne de caractères
- "toto" est une instance de string

Types et instances : notations

Sémantique d'un type `type`

$$[\![type]\!]$$

Valeur `val` **instance du type** `type`

$$val :: type$$

Types de base

- Types des données atomiques
  (non décomposables, *i*.e., diffèrentes des tableaux et d'autres structures)

- Instances : valeurs de base : entier, chaînes de caractères, booléens, etc

Dans ce cours on considère qu'il existe une liste fixée de types de base:

bool, int, float, string, date

Types des structures

Trois sortes de structures :

- Les *records* (équivalent à `struct`)
- Les *dicts* (dictionnaires)
- Les *arrays* (tableaux)

Des constructions de types différentes pour chacune

Records

Type = ensemble de couples (champ, type)

Syntaxe

$$<\text{champ}_1 : type_1, \ldots, \text{champ}_n : type_n>$$

où l'ordre des champs n'est pas important

Sémantique

Tous les objets qui vérifient les conditions suivantes :

Records : exemples

$$< a : int, b : string >$$

- { "a" : 3, "b" : "toto" } instance
- { "a" : 3 } pas instance
  il manque le champ b
- { "a" : 3, "b" : "toto", "c" : true } instance
  le type ne contraint pas le champ c
- { "a" : " titi ", "b" : "toto" } pas instance
  la valeur du champ a n'est pas une instance de *int*

Dictionnaires

Syntaxe

$\{type\}$ ou bien dict($type$)

Sémantique

Tous les objets qui vérifient les conditions suivantes :

Dictionnaires : exemples

$$\{string\}$$

- { "a" : " titi ", "b" : "toto" } instance
- { "a" : 3, "b" : "toto" } pas instance
  le champ a n'a pas le bon type

$$\{< a : int >\}$$

- { "c" : { "a" : 3, "b" : "toto" } } instance
- { "c" : { "b" : "toto" } } pas instance
  le champ a n'est pas défini dans la sous-structure associée au champ c

Tableaux

Syntaxe

$[type]$ ou bien array($type$)

Sémantique

Tous les objets qui vérifient les conditions suivantes :

Tableaux : exemples

$$[string]$$

- [" titi ", "toto"] instance
- [3, "toto"] pas instance
  la case 0 n'a pas le bon type

Multiplicité des types

```
val = {"a":"toto","b":"titi"}
```

- *{string}*
- *< a : string, b : string >*
- *< a : string >*
- *...*

Types vs JSON Schema

Plus d'expressivité côté JSON-Schema :

- champs optionels ou obligatoires
- regexs de clés ! type pour les dictionnaires
- gestion des nulls
- ...

$< champ_1 : \tau_1, \ldots, champ_n : \tau_n >$

$[\tau]$

```
{
  "type": "object",
  "properties": {
      "champ1": { /* traduction de τ1 */ },
      ...
      "champn": { /* traduction de τn */ }
  },
  "required": [ "champ1", ..., "champn" ]
}
```

```
{
  "type": "array",
  "items": { /* traduction de τ */ }
}
```

Modèle de données orienté document

Modèle de données

Un modèle de données définit un mode de représentation de l'information:

- Un mode de représentation des données, via un **LDD**
- Un mode de représentation des contraintes sur ces données, via le **LDD**
- Un ensemble d'opérations (**CRUD**) pour manipuler les données, via le **LMD**

Pour le modèle de données orienté document :

- Une grammaire pour représenter un document
- Deux opérations essentielles pour manipuler les documents, la sélection et la projection

---

LDD : Langage de Définition de Données (CREATE, ALTER, DROP, TRUNCATE, COMMENT, RENAME)
CRUD : Create, Read, Update, Delete
LMD : Langage de Manipulation de Données (Ex., en SQL: SELECT, INSERT, UPDATE, DELETE, MERGE …)

Grammaire d'un document

```
<document> := <objet>
<objet> := {<clé-valeur> (, <clé-valeur>)* } | {}
<clé-valeur> := <string>:<valeur>
<valeur> := <atome> | <liste> | <objet>
<atome> := <string> | <number> | <boolean> | ...
<liste> := [ <valeur> (, <valeur>)* ] | []
```

Un document est un objet contenant des paires clé-valeur. Une valeur peut être une instance d'un type de base, une liste ou un objet

Grammaire d'un document – Exemple 1

```
<document> := <objet>
<objet> := {<clé-valeur> (, <clé-valeur>)* } | {}
<clé-valeur> := <string>:<valeur>
<valeur> := <atome> | <liste> | <objet>
<atome> := <string> | <number> | <boolean> | ...
<liste> := [ <valeur> (, <valeur>)* ] | []
```

```
{
  w: 1,
  x: "vx",
  y: ["vy1", "vy2", 3],
  z: {
    zx: "vzx",
    zy: ["vzy1", "vzy2"]
  }
}
```

Grammaire d'un document – Exemple 2

```
<document> := <objet>
<objet> := {<clé-valeur> (, <clé-valeur>)* } | {}
<clé-valeur> := <string>:<valeur>
<valeur> := <atome> | <liste> | <objet>
<atome> := <string> | <number> | <boolean> | ...
<liste> := [ <valeur> (, <valeur>)* ] | []
```

Pourquoi ce document n'est-il pas conforme à la grammaire (6 erreurs) ?

```
{
  w: 1,
  x: vx,
  y: [vy: "vy1"],
  3: {
    zx: {"vzx"},
    zx: ["vzx1"]
  }
}
```

Grammaire pour la sélection

Soit une collection de documents C, un langage de manipulation permet de sélectionner les documents de C qui satisfont une condition donnée (pour une interrogation, une mise à jour ou une suppression)

```
<condition> := <condition> opérateur <condition> |
               opérateur(<condition>) |
               {op_expression: {<expression>}} |
               <clé-valeur> |
               <clé>: {opérateur: <valeur>}
```

Grammaire pour représenter une sélection (condition de filtre sur les documents)

Une condition peut être simplement une paire clé-valeur, éventuellement supportée par un opérateur. Plusieurs conditions peuvent être combinées par un opérateur binaire

Grammaire pour la sélection

```
{
  w: 1,
  x: "vx",
  y: ["vy1", "vy2", 3],
  z: {
    zx: "vzx",
    zy: ["vzy1", "vzy2"]
  }
}
```

Parmi les conditions (formelles) ci-dessous, lesquelles permettent de retourner le document ci-contre ?

```
x: "vx"
```
✓

```
w: {>: 5}
```
✗

```
non(x: "vx")
```
✗

Grammaire pour la sélection - expressions

Une expression est une représentation de haut niveau permettant :

- D'exprimer les autres conditions
- De comparer 2 champs d'un même document

Une expression peut inclure :

- Un littéral (e.g., 1, "valeur")
- Un chemin vers un champ, préfixé par un symbole (e.g., "`$champ1.champ2`")
- Un objet expression `{<cle>: <expression>, ... }`
- Un opérateur d'expression (similaire à une fonction)

  `{opérateur: [argument1, argument2, ... ] }`

Grammaire pour la sélection - exemple d'expression

```
<expression> := <atome> |
        $<clé> |
        <clé>: <expression> |
        {opérateur : [<clé> | <valeur>, ...]}
```

Grammaire pour représenter une expression

```
w: {"<": 5}

{op_expression : {< : [$w, 5]}}
```

Deux conditions équivalentes (la seconde avec une expression)

```
{op_expression : {= : ["$x", "$z.zx"]} }
```

Une condition utilisant une expression pour comparer 2 champs d'un même document

Grammaire pour la projection

Soit une collection de documents, un langage de manipulation permet de spécifier les champs (attributs) qui seront mis dans les documents résultats (pour une interrogation ou une mise à jour).

```
<projection> := <projection>, <projection> |
                <clé-valeur> |
                <clé>: {opérateur: <valeur>}
```

Grammaire pour représenter une projection. Elle consiste essentiellement en une liste de paires clé-valeur, éventuellement supportées par un opérateur

---

Dans MongoDB, la projection consiste surtout à spécifier les champs des documents résultats au moyen d'une valeur booléenne

Grammaire pour la projection - exemple

```
# champs w et x dans les documents résultats
w: 1, x: true

# tous les champs dans les documents résultats, sauf w et zv
w: 0, z: 0

# nouveau champ moy_w dans les documents résultats
moy_w: {moyenne: w}

# champs x, z et nouveau champ t dans les documents résultats
x: 1, z: 1, t: {taille: z}
```

Exemples de projection simple, avec opérateur et composée de plusieurs éléments

Grammaire pour la projection - exemple

```
# champs w et x dans les documents résultats
w: 1, x: true

# tous les champs dans les documents résultats, sauf w et zv
w: 0, z: 0

# nouveau champ moy_w dans les documents résultats
moy_w: {moyenne: w}

# champs x, z et nouveau champ t dans les documents résultats
x: 1, z: 1, t: {taille: z}
```

Exemples de projection simple, avec opérateur et composée de plusieurs éléments

Grammaire pour la projection - exemple

```
# champs w et x dans les documents résultats
w: 1, x: true

# tous les champs dans les documents résultats, sauf w et zv
w: 0, z: 0

# nouveau champ moy_w dans les documents résultats
moy_w: {moyenne: w}

# champs x, z et nouveau champ t dans les documents résultats
x: 1, z: 1, t: {taille: z}
```

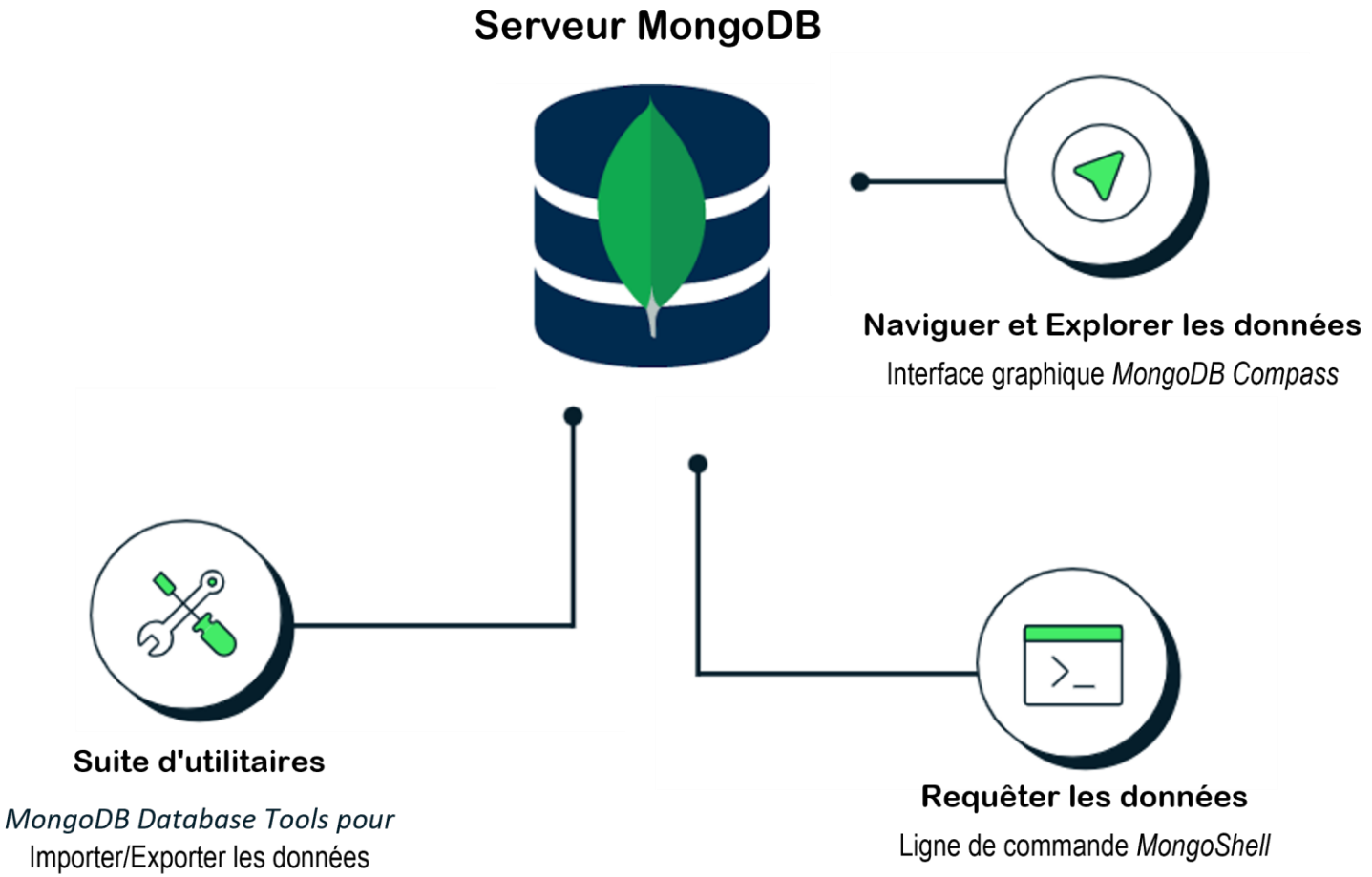
Exemples de projection simple, avec opérateur et composée de plusieurs éléments

Caractéristiques de MongoDB

Généralités

- SGBD orienté documents
- Open-source
- Populaire (5ème SGBD le plus utilisé)
- Passage à l'échelle horizontal (sharding) et réplication
- Système CP (cohérent et résistant au morcellement)
- "strong consistency" (lectures sur serveur primaire)
- "eventual consistency" (lectures sur différents serveurs)
- Traitements distribués (Map Reduce, aggregation pipeline)
- GUI (Compass, Robot 3T) et nombreux drivers

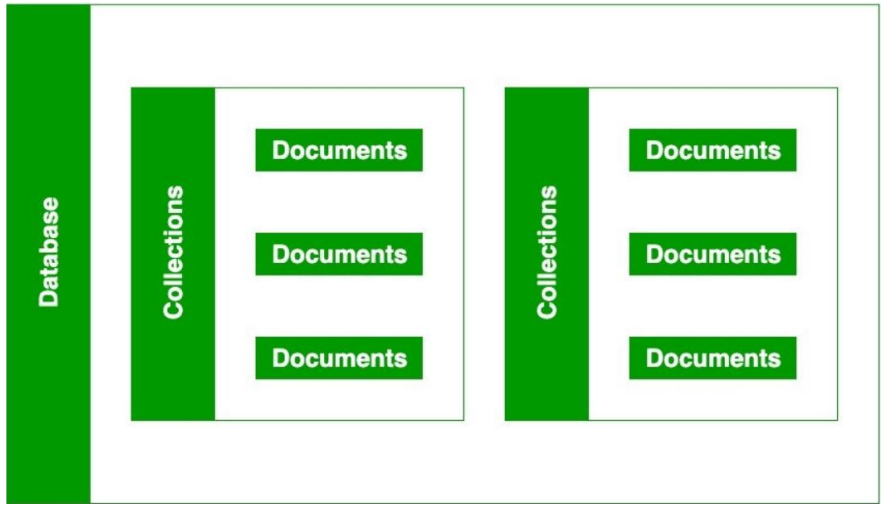
Architecture



Concepts principaux - BD et collection

Base de données :

- Ensemble de collections
- Espace de stockage



Collection (table en modèle relationnel) :

- Ensemble de documents qui partagent un objectif ou des similarités
- Pas de ”schéma” prédéfini

Concepts principaux - document

Document :

- Un enregistrement dans une collection
- Syntaxe et stockage au format BSON
- Identifiant d'un document (clé "*_id*")

BSON = "**Binary JSON**" (JavaScript Object Notation) avec améliorations :

- Ensemble de paires clé/valeur
- Une valeur peut être un objet complexe (liste, document,
- ensemble de valeurs, etc.)
- Représentation de nouveaux types (e.g., dates)
- Facilité de parsing (e.g., entiers stockés sur 32/64 bits)

Concepts principaux - document

Syntaxe d'un document en MongoDB (format BSON)

- _id est un identifiant (généré ou manuel)
- att-1 est une clé dont la valeur est une chaîne de caractères
- att-2 est une clé dont la valeur est un nombre
- att-3 est une clé dont la valeur est une liste de valeurs
- att-k est une clé dont la valeur est un document inclus

```
{
_id : <identifiant>,
"att-1" : "val-1",
"att-2" : val-2,
"att-3" : ["val-31", "val-32", ...]
...
"att-k" : {
            "att-k1" : "val-k1",
            ...
          }
}
```

Relations inter-documents

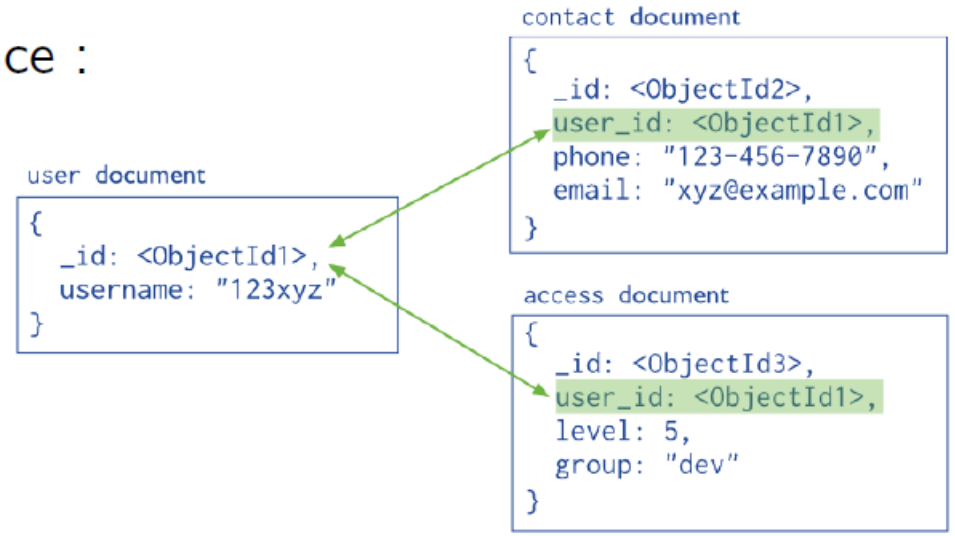
Relation entre les documents de différentes collections :

- Par référence : l'identifiant d'un document (son "_id") est utilisé comme valeur attributaire dans un autre document
  - se rapproche du modèle de données normalisées
  - nécessite des requêtes supplémentaires côté applicatif

- Par inclusion ("embedded") : un "sous-document" est utilisé comme valeur
  - philosophie "non-relationnelle" (pas de jointure)
  - meilleures performances en lecture
  - redondance possible

Relations inter-documents

Relation par référence :



Relation par inclusion (embedded) :

Modélisation d'une BD

Considérations pour modéliser une BD au niveau physique :

- Accès par une clé (identifiant d'un document)
- Schéma optionnel → ajout d'un champ à tout moment
- Pas de jointure → redondances possibles
- Inclusion → moins de lectures
- Opérations atomiques sur un seul document, mais pas sur plusieurs → état incohérent temporaire (souvent tolérable)

« *Application-driven* » : les besoins applicatifs guident la conception pour identifier, stocker et accéder aux concepts (types de requêtes, ratio lecture/écriture, croissance du nombre de documents)

Modélisation d'une BD - recommandations

Relation par **inclusion** pour :

- les entités fréquemment lues ensemble (article/commentaires) une entité dépendante d'une autre (facture/client)
- des données mises à jour en même temps (nombre d'exemplaires d'un livre et informations sur les emprunteuses)
- besoin de rechercher des documents via un index multi-clés(catégories d'un livre)

Relation par **référence** pour :

- éviter une forte redondance sans "gain" (livre/éditeur)
- des entités fortement connectées "many to many" (livres/auteurs)
- des données hiérarchiques (catégories/sous-catégories/…)

Requêtage avec MongoDB

CRUD = IFUD

Toute opération sur un seul document est atomique :

- `insert`
- `find`
- `update`
- `delete`

Lors des insertions et mises à jour, la base de données et la collection sont automatiquement créées si elles n'existent pas

Jeu de données

```
{ _id: "Ana",
  annee: 2020,
  groupes: ["A", "A1"],
  notes: [{ue: "BD", note: 17},
          {ue: "WEB", note: 18}]
}
{ _id: "Bob",
  annee: 2022,
  groupes: ["A", "A2"],
  notes: [{ue: "BD", note: 19},
          {ue: "WEB", note: 14}]
}
{ _id: "Cya",
  annee: 2021,
  groupes: ["A", "A1"],
  notes: [{ue: "BD", note: 14}]
}
{ _id: "Dan",
  annee: 2020,
  groupes: ["B", "B1"],
  notes: [{ue: "BD", note: 9},
          {ue: "WEB", note: 16}]
}
```

Ana

annee : 2020
groupes : [A, A1]
notes : [
{ue : 'BD', note : 17},
{ue : 'WEB', note : 18}]

Bob

annee : 2022
groupes : [A, A2]
notes : [
{{ue : 'BD', note : 19},
{ue : 'WEB', note : 14}]

Cya

annee : 2021
groupes : [A, A1]
notes : [
{{ue : 'BD', note : 14}]

Dan

annee : 2020
groupes : [B, B1]
notes : [
{{ue : 'BD', note : 9},
{ue : 'WEB', note : 16}]

*Collection **etudiants** = **etu** contenant 4 documents représentant chacun un.e Étudiant.e, son année d'inscription, ses groupes et ses notes*

Interrogation

`db.collection.find(<condition>, <projection>)`

- Le document `<condition>` spécifie les critères pour sélectionner les documents pertinents
- Le document facultatif `<projection>` indique les champs à inclure dans les documents résultats :
- si non spécifié, retourne tous les champs
- identifiant `_id` inclus par défaut
- `<clé> : 1 | true` (inclusion) ou `<clé> : 0 | false` (exclusion)
- pour les champs de documents imbriqués, `<clé>.<sous-clé>: 1` ou `<clé>: {<sous-clé> : 1}`

Interrogation - exemples de projection

```
db.etu.find()
```

Ana
annee : 2020
groupes : [A, A1]
notes : [
{ue : 'BD', note : 17},
{ue : 'WEB', note : 18}]

Bob
annee : 2022
groupes : [A, A2]
notes : [
{{ue : 'BD', note : 19},
{ue : 'WEB', note : 14}]

Cya
annee : 2021
groupes : [A, A1]
notes : [
{{ue : 'BD', note : 14}]

Dan
annee : 2020
groupes : [B, B1]
notes : [
{{ue : 'BD', note : 9},
{ue : 'WEB', note : 16}]

```
db.etu.find({},
  {
    annee: 1,
    groupes: true
  }
)
```

Ana
annee : 2020
groupes : [A, A1]

Bob
annee : 2022
groupes : [A, A2]

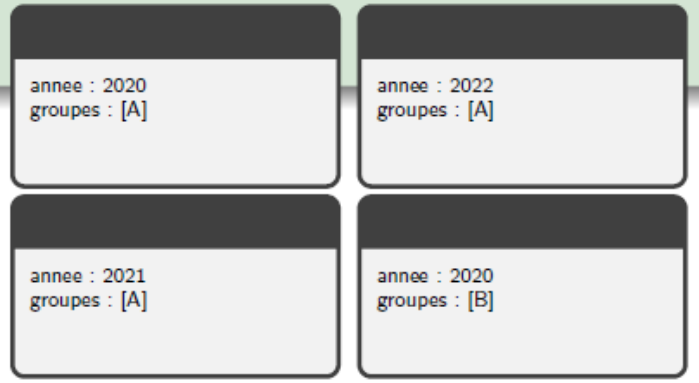
Cya
annee : 2021
groupes : [A, A1]

Dan
annee : 2020
groupes : [B, B1]

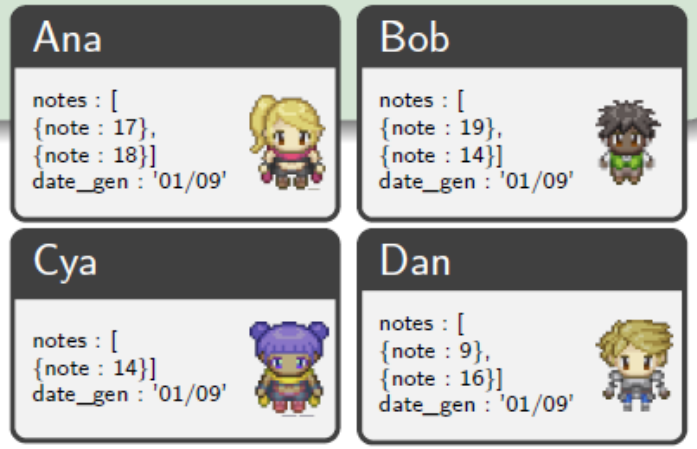
Exemples d'interrogation qui retourne tous les documents avec tous les champs (gauche) et seulement l'année et les groupes (droite)

Interrogation - exemples de projection

```
db.etu.find({},
  {
    _id: 0,
    notes: 0,
    groupes: {$slice: 1}
  }
)
```




```
db.etu.find({},
  {
    "notes.note": 1,
    date_gen: "01/09"
  }
)
```



Exemples qui retournent tous les documents, sans l'identifiant ni les notes et seulement le premier élément de groupes (gauche), avec le sous-champ note et un nouveau champ (droite)

Interrogation - opérateurs

Rappel de la grammaire :

```
<condition> := <condition> opérateur <condition> |
               opérateur(<condition>) |
               {op_expression: {<expression>}} |
               <clé-valeur> |
               <clé>: {opérateur: <valeur>}
```

Type d'opérateurs :

- comparatif `$eq`, `$ne`, `$gt`, `$gte`, `$lt`, `$lte`, `$in`, `$nin`
- logique `$and`, `$or`, `$not`, ...
- élément `$exists` (présence d'un champ), `$type`
- évaluation `$expr`, `$regex`, `$text`, `$jsonSchema`, ...
- tableau `$all`, `$elemMatch`, `$size`
- autres (géospatial, etc.)

Interrogation - exemples de comparaison

```
db.etu.find({
    annee: 2022
  }
)
```

Bob

annee : 2022
groupes : [A, A2]
notes : [
{{ue : 'BD', note : 19},
{ue : 'WEB', note : 14}]

```
db.etu.find({
    annee: {$gt: 2020}
  },
  {
    notes: 0,
  }
)
```

Cya

annee : 2021
groupes : [A, A1]

Bob

annee : 2022
groupes : [A, A2]

```
{ _id: "Ana",
  annee: 2020,
  groupes: ["A", "A1"],
  notes: [{ue: "BD", note: 17},
          {ue: "WEB", note: 18}]
}
{ _id: "Bob",
  annee: 2022,
  groupes: ["A", "A2"],
  notes: [{ue: "BD", note: 19},
          {ue: "WEB", note: 14}]
}
{ _id: "Cya",
  annee: 2021,
  groupes: ["A", "A1"],
  notes: [{ue: "BD", note: 14}]
}
{ _id: "Dan",
  annee: 2020,
  groupes: ["B", "B1"],
  notes: [{ue: "BD", note: 9},
          {ue: "WEB", note: 16}]
}
```

Exemples qui retournent les documents de 2022 (gauche), et ceux avec une année supérieure à 2020, sans les notes (droite)

Interrogation - exemples avec $expr

Opérateur générique $expr :

- Fonctions supplémentaires (dates, maths, etc.)
- Comparaison de 2 champs d'un même document
- Accès aux champs par des chemins **$cle**

```
db.etu.find({
    $expr: { $eq: [ $annee,
        2022 ]}
  }
)
```

```
db.etu.find({
    $expr: { $gt: [ $annee,
        2020 ] }
  },
  { notes : 0 }
)
```

Mêmes exemples que la diapositive précédente, mais avec l'opérateur $expr

Interrogation - exemples avancés

```
db.etu.find({
  notes: { $elemMatch: {
         ue: "WEB",
         note: { $gte: 15 }
         }
       }
})
```


```
db.etu.find({
  $or: [
    { groupes : { $in: ["A1",
        "B2"] }},
    { _id: { $regex: /A/i } }}
  ]
})
```

Exemples qui retournent les documents avec une note WEB > 15 (gauche), et ceux avec un groupe soit A1 soit B2 ou avec un _id contenant un « a » (droite)

Pour plus de détails sur le regex: https://www.mongodb.com/docs/manual/reference/operator/query/regex/

Interrogation - exemples avancés

Insertion d'une nouvelle étudiante : Zoé

```
db.etu.insertOne({
    _id: "Zoé",
    annee: 2022,
    annee_premiere_insc: 2022,
    cursus_precedent: {
      etablissement: 'XYZ',
      diplome: true
    }
})
```

Zoé

annee : 2022
annee_premiere_insc : 2022,
cursus_precedent : {
etablissement : 'XYZ',
diplome : true}

```
db.etu.find({
    cursus_precedent.
        diplome: {
      $not: { $eq: true }
    }
})
```

Ana

annee : 2020
groupes : [A, A1]
notes : [
{ue : 'BD', note : 17},
{ue : 'WEB', note : 18}]

Bob

annee : 2022
groupes : [A, A2]
notes : [
{ue : 'BD', note : 19},
{ue : 'WEB', note : 14}]

Cya

annee : 2021
groupes : [A, A1]
notes : [
{ue : 'BD', note : 14}]

Dan

annee : 2020
groupes : [B, B1]
notes : [
{ue : 'BD', note : 9},
{ue : 'WEB', note : 16}]

```
db.etu.find({
    $expr: {
      $eq: [ $annee,
          $annee_premiere_insc
          ]
    }
})
```

Zoé

annee : 2022
annee_premiere_insc : 2022,
cursus_precedent : {
etablissement : 'XYZ',
diplome : true}

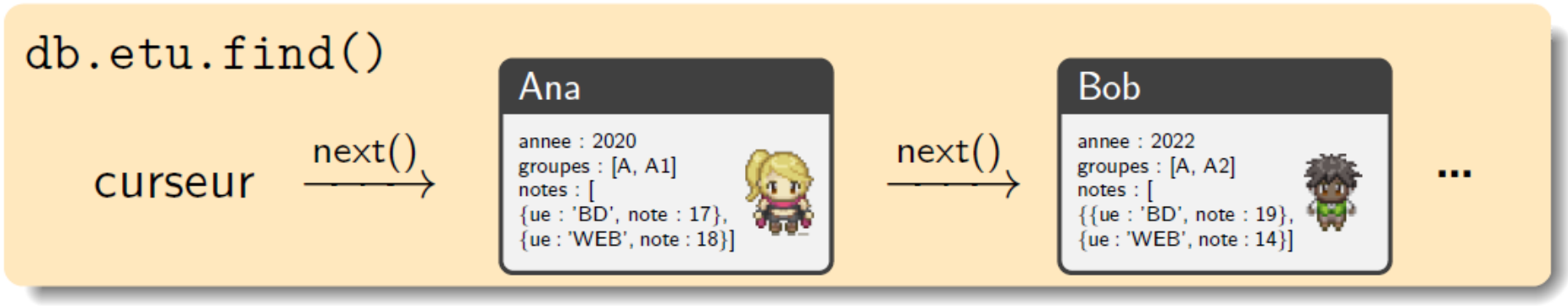
Exemples qui retournent les documents qui n'ont pas de précédent diplôme (gauche), et ceux nouvellement arrivés (droite)

Interrogation - curseur résultat

L'opérateur `find()` retourne un curseur, donc besoin d'itérer dessus pour parcourir chaque document résultat

Quelques méthodes sur le curseur (pour *mongo shell*) :

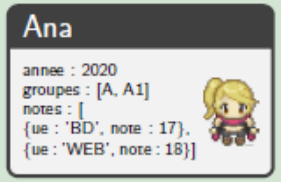
- `hasNext(), next()`
- `count(), limit(<number>), skip(<number>)`
- `sort({ <clé-valeur> (, <clé-valeur>)* })`
- `forEach(f)` exécute une fonction JS sur chaque résultat



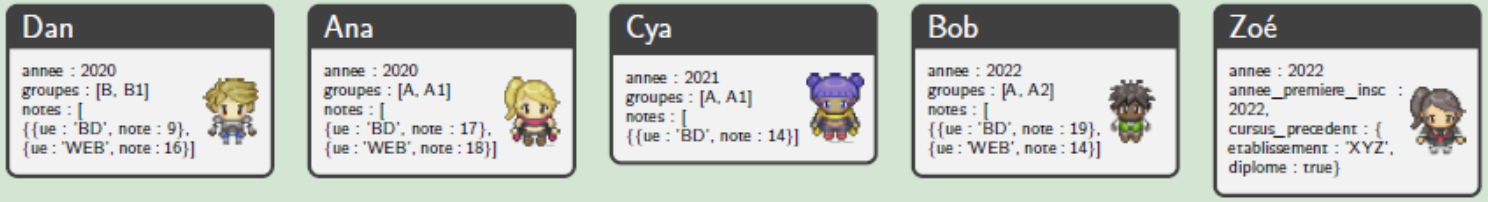
Pour plus de détails sur les curseur: https://www.mongodb.com/docs/manual/reference/method/#std-label-js-query-cursor-methods

Interrogation - exemples curseur

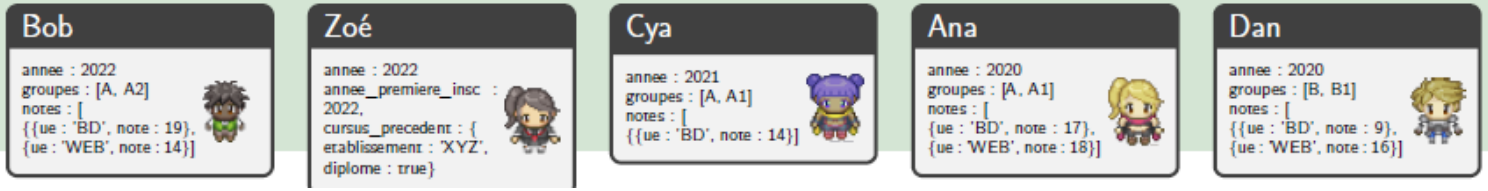
```
# nombre de documents dans la collection etu
db.etu.find().count()    # 5

# un document aléatoire, équivalent à db.etu.findOne() en mongo
    shell
db.etu.find().limit(1)
```




```
# tous les documents triés par année croissante (tri non
    cohérent entre 2 exécutions si documents avec même année)
db.etu.find().sort({annee: 1})
```




```
# tous les documents triés par année décroissante puis par _id
    croissant (tri cohérent grâce au _id)
db.etu.find().sort({annee: -1, _id: 1})
```



Insertion

db.collection.insertOne(<document>, *options*)

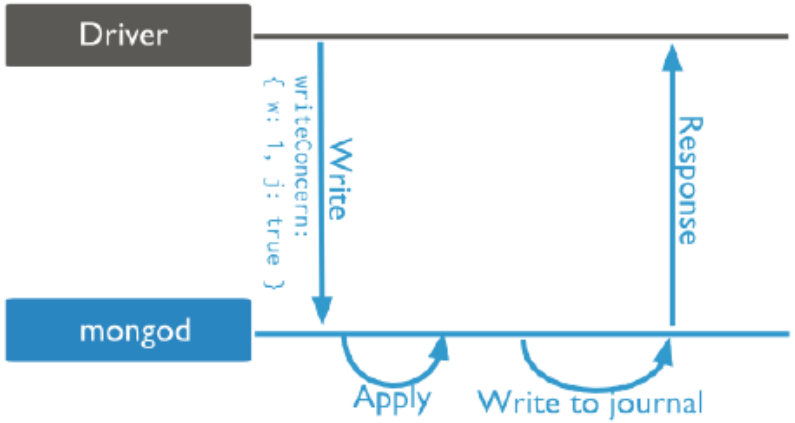
db.collection.insertMany([<document>, <document>, ...],*options*)

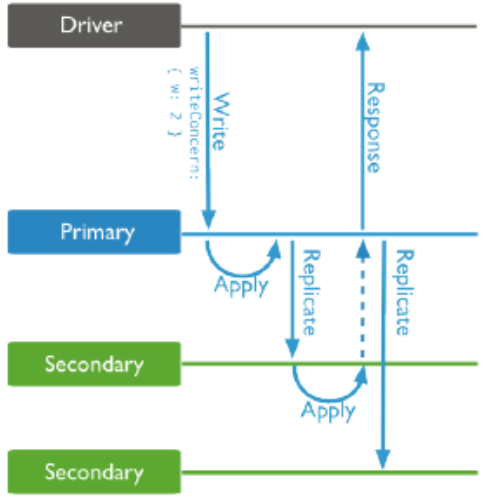
- Les instances de `<document>` sont les documents à insérer
- Identifiant `_id` des documents à insérer :
  - soit spécifié dans le document (s'assurer de l'unicité !)
  - soit généré par MongoDB (type ObjectID sur 12 octets)
- Retourne un document contenant une confirmation et le(s) identifiant(s) des documents insérés

https://www.mongodb.com/docs/manual/reference/method/db.collection.insertOne/
https://www.mongodb.com/docs/manual/reference/method/db.collection.insertMany/
https://www.mongodb.com/docs/manual/reference/method/ObjectId/

Insertion - garantie de succès

- Un second document facultatif *options* pour spécifier la garantie de succès de l'opération pouvant contenir :
  - w, nombre de confirmations d'écriture (e.g., 0, 1, "majority")
  - *j*, un booléen pour écrire dans le journal
  - *wtimeout*, une limite de temps en ms





Exemples de writeConcern : "journal" et "2 écritures"

Insertion - exemple

```
db.etu.insertOne({
    _id: "Ela",
    annee: 2022,
    groupes: ["B", "B1"]
  },
  {
    writeConcern: {
      j: true
  }
)
```

```
db.etu.insertMany([{
    _id: "Flo",
    annee: 2022,
    groupes: ["B", "B2"]
  },
  {
    _id: "Gad",
    annee: 2022
  }
])
```

Exemples d'une insertion d'un document avec le paramètre optionnel imposant l'écriture dans le journal avant confirmation (gauche) et d'une insertion de plusieurs documents (droite)

Insertion - exemple

```
db.etu.insertOne({
    _id: "Ela",
    annee: 2022,
    groupes: ["B", "B1"]
  },
  {
    writeConcern: {
      j: true
  }
)
```

```
{
   acknowledged : true,
   insertedId : "Ela"
}
```

```
1  db.etu.insertMany([{
2      _id: "Flo",
3      annee: 2022,
4      groupes: ["B", "B2"]
5    },
6    {
7      _id: "Gad",
8      annee: 2022
9    }
10 ])
```

```
{
   acknowledged : true,
   insertedIds : ["Flo", "Gad"]
}
```

Exemples d'une insertion d'un document avec le paramètre optionnel imposant l'écriture dans le journal avant confirmation (gauche) et d'une insertion de plusieurs documents (droite)

Mise à jour

db.collection.updateOne(*<condition>, update, options***)**

db.collection.updateMany(*<condition>, update, options***)**

- Un document `<condition>` spécifie les critères pour sélectionner les documents à mettre à jour
- Un document update spécifie les opérations de mise à jour
- Un document options facultatif avec :
  - upsert : `<booléen>` (création d'un document si aucun résultat n'est retourné par <*condition*>)
  - *writeConcern* : <document> (garantie d'écriture)

https://www.mongodb.com/docs/manual/reference/method/db.collection.updateOne/
https://www.mongodb.com/docs/manual/reference/method/db.collection.updateMany/

Mise à jour - exemple

- Retourne un document contenant une confirmation et le nombre de documents trouvés et modifiés

```
db.etu.updateOne(
  { _id: "Cya" },
  { $push: { notes : { ue: "
      WEB", note: 11 }}}
)
```

Exemples d'une mise à jour de la note WEB de Cya

```
db.etu.updateMany(
  { $not: { 'notes.note': {
      $lt: 10 }}},
  { $set: { "ECTS" : 3 }}
)
```

Mise à jour de tous les documents sans note inférieure à 10 par l'ajout du nombre d'ECTS acquis

https://www.mongodb.com/docs/manual/reference/operator/update/

Mise à jour - exemple

- Retourne un document contenant une confirmation et le nombre de documents trouvés et modifiés

```
db.etu.updateOne(
  { _id: "Cya" },
  { $push: { notes : { ue: "
      WEB", note: 11 }}}
)
```

```
{
  acknowledged: true,
  matchedCount: 1,
  modifiedCount: 1
}
```

Exemples d'une mise à jour de la note WEB de Cya

```
db.etu.updateMany(
  { $not: { 'notes.note': {
      $lt: 10 }}},
  { $set: { "ECTS" : 3 }}
)
```

```
{
  acknowledged: true,
  matchedCount: 3,
  modifiedCount: 3
}
```

Mise à jour de tous les documents sans note inférieure à 10 par l'ajout du nombre d'ECTS acquis

https://www.mongodb.com/docs/manual/reference/operator/update/

Suppression

db.collection.deleteOne(*<condition>, options***)**

db.collection.deleteMany(*<condition>, options***)**

```
db.etu.deleteOne(
  { _id: "Dan" },
  { writeConcern: { w: 2 }}
)
```

Suppression du document d'identifiant Dan avec garantie d'écriture

```
db.etu.deleteMany(
  { $not: {annee: 2022 }}
)
```

Suppression des documents dont la date n'est pas 2022

Suppression

db.collection.deleteOne(*<condition>, options***)**

db.collection.deleteMany(*<condition>, options***)**

```
db.etu.deleteOne(
  { _id: "Dan" },
  { writeConcern: { w: 2 }}
)
```

```
{
  acknowledged: true,
  deletedCount: 1
}
```

Suppression du document d'identifiant Dan avec garantie d'écriture

```
db.etu.deleteMany(
  { $not: {annee: 2022 }}
)
```

```
{
  acknowledged: true,
  deletedCount: 3
}
```

Suppression des documents dont la date n'est pas 2022

Pipelines d'agrégation

Agrégation

Définition

L'agrégation est le processus qui consiste à passer par différentes étapes avec une vaste collection de documents pour les traiter. Ces étapes sont appelées "pipeline".

- Il est bien évidemment possible de réaliser des calculs d'agrégats de type:
  - Somme
  - Moyenne
  - Minimum
  - Maximum
- La fonction qui permet de le faire est aggregate()

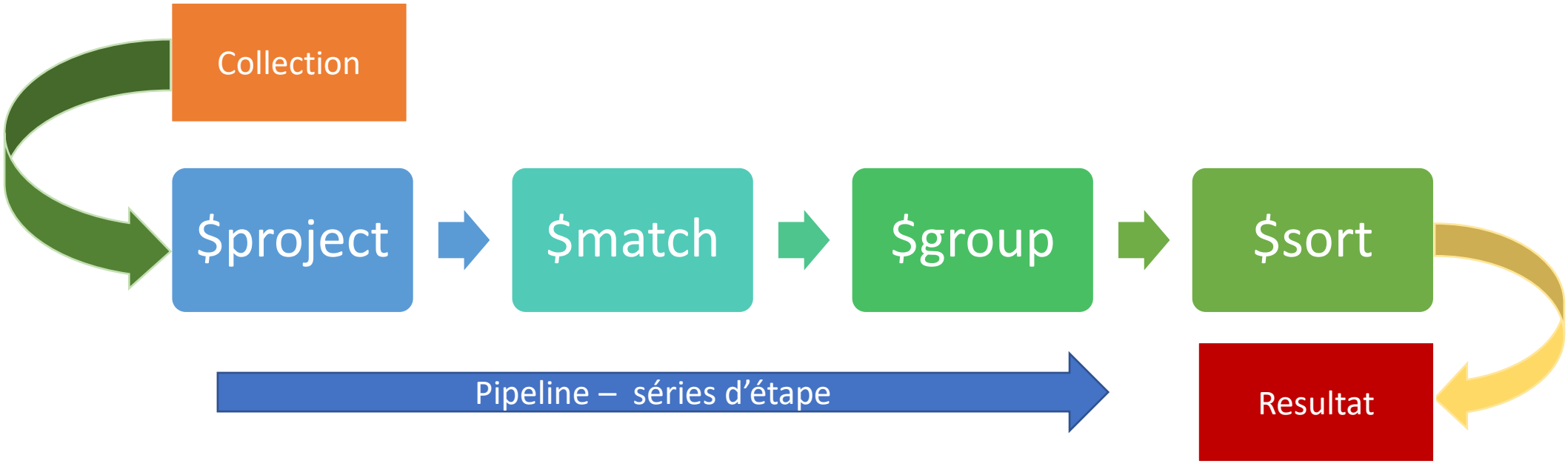
Agrégation - Aventage

- L'avantage avec la fonction aggregate() est d'effectuer des opérations de traitement de données complexes sur les documents d'une collection et de produire des résultats plus avancés telles que: *le regroupement, la filtration, la projection, et les calculs mathématiques, au sein d'un seul appel*

- Par rapport aux requêtes simples :
  - Permet d'avoir plus de flexibilité et évite d'avoir à effectuer plusieurs requêtes simples
  - Possibilité de Créer des Vues Virtuelles qui agrègent les données de plusieurs collections de manière significative
  - Offre un meilleur traitement pour les Tableaux et Documents Emboîtés avec une flexibilité accrue pour travailler avec des données complexes et hiérarchiques

Pipeline

- La fonction aggregate() prends en paramètre un tableau nommé `pipeline`: au composé d'une suite d'opérations.
- Fonctionnement du pipeline d'agrégation de MongoDB



Opérations de Pipeline Courantes

- `$match` - Filtre les documents pour ne transmettre à l'étape suivante du pipeline que les documents qui correspondent aux conditions spécifiées.

```
db.users.aggregate(
    [ { $match : { email : "example@email.com" } } ]
);
```

Opérations de Pipeline Courantes

- `$group` – Permet de regrouper les documents d'une collection sur la base d'une "*clé de groupe*".

```
db.sales.aggregate( [
    {
      $group: {
         _id: null,
         count: { $count: { } }
      }
    }
  ] )
```

Opérations de Pipeline Courantes

- `$sort` – Cette étape permet de réorganiser le document résultant (Ascendant 1 ou Descendant -1)

```
db.contestants.aggregate(
    [
      { $sort : { votes : 1 } }
    ]
 )
```

Opérations de Pipeline Courantes

- `$project` – Il est utilisé pour sélectionner certains champs spécifiques de la collection résultante à envoyer au client

```
db.events.aggregate(
    [
        $project: {
            _id: 0,
            name: 1,
            description: 1,
            banner: 1,
            opening_date: 1,
            closing_date: 1,
        },
    ]
 )
```

Note : *La valeur 1 attribuée à un champ signifie qu'il est inclus. Ce champ sera inclus dans le document de sortie. 0 signifie l'inverse. Le champ sera exclu du document.*

Il est important de noter que les valeurs non nulles sont également considérées comme inclusives

Opérations de Regroupement Avancées

- `$unwind` – Déconstruit un champ de tableau à partir des documents d'entrée afin de produire un document pour chaque élément. Chaque document de sortie est le document d'entrée avec la valeur du champ du tableau remplacée par l'élément.

```
db.events.aggregate( [ { $unwind : "$organizer" } ] )
```

Opérations de Pipeline Courantes

- `$lookup` – Effectue une jointure externe gauche avec une collection dans la même base de données pour filtrer les documents de la collection "jointe" en vue de leur traitement.

```
db.orders.aggregate( [
    {
      $lookup:
        {
          from: "inventory",
          localField: "item",
          foreignField: "sku",
          as: "inventory_docs"
        }
   }
 ] )
```

Opérations de Regroupement Avancées

- `$facet` – Traite plusieurs lignes d'agrégation en une seule étape sur le même ensemble de documents d'entrée. Chaque sous-pipeline a son propre champ dans le document de sortie où ses résultats sont stockés sous la forme d'un tableau de documents.

```
db.artwork.aggregate( [
    {
      $facet: {
        "categorizedByTags": [
          { $unwind: "$tags" },
          { $sortByCount: "$tags" }
        ],
        "categorizedByPrice": [
          { $match: { price: { $exists: 1 } } },
          {
            $bucket: {
              groupBy: "$price",
            }
```

Bravo! Vous avez réussi à tout lire ☺