

MIF24 - SGBD orientés document (MongoDB)

Fabien Duchateau

fabien.duchateau [at] univ-lyon1.fr

Université Claude Bernard Lyon 1

2024 - 2025



<https://forge.univ-lyon1.fr/mif24-bdnosql/mif24-bdnosql/>

Depuis les années 1970, dominance du modèle relationnel

Émergence du web et du phénomène "Big Data" :

- ▶ Grandes plateformes ou applications web gérant des millions d'utilisateur·rice·s et/ou d'objets
- ▶ Explosion du volume de données à stocker et à traiter
- ▶ Données de plus en plus complexes et hétérogènes

Contexte

Depuis les années 1970, dominance du modèle relationnel

Émergence du web et du phénomène "Big Data" :

- ▶ Grandes plateformes ou applications web gérant des millions d'utilisateur·rice·s et/ou d'objets
- ▶ Explosion du volume de données à stocker et à traiter
- ▶ Données de plus en plus complexes et hétérogènes

Limites des SGBD relationnels (utilisant le langage SQL) pour ces nouveaux usages, à cause du mécanisme de jointures, des contraintes d'intégrité et des transactions

Le Big Data

Big Data : modélisation, stockage et analyse d'un ensemble de données volumineuses, croissantes et hétérogènes, dont l'exploitation permet la prise de décisions ou la découverte de nouvelles connaissances

Les "3V", caractéristiques du Big Data :

- ▶ **Volume** (e.g., plusieurs zettaoctets/an générés sur le web)
- ▶ **Vélocité** ou fréquence de génération des données, (e.g., 4000 To/jour pour Facebook en 2016 ou 7000 To/seconde pour le radiotélescope Square Kilometre Array)
- ▶ **Variété** ou hétérogénéité (e.g., images, texte, données géo-démographiques)

Extension à "5V" avec véracité (provenance) et valeur (ajoutée)

http://fr.wikipedia.org/wiki/Big_data

Exemples d'application du Big Data

- ▶ Création de cartes de navigation par Fontaine Maury, au 19^{ème} siècle, à partir de vieux journaux de bord (précurseur)
- ▶ Large Hadron Collider (LHC), un accélérateur de particules
- ▶ Décodage du génome humain
- ▶ Programmes de surveillance (e.g., Prism)
- ▶ Réduction de 22% du gaspillage alimentaire
- ▶ Découverte d'un effet secondaire dû à la prise de deux médicaments par analyse des requêtes des internautes (Yahoo)
- ▶ Confirmation de la censure par analyse de la fréquence de noms de personnes (Chagall en Allemagne, Trotski en URSS)



Motivation

Distribution des données sur différents serveurs et "data centers" :

- ▶ Passage d'un système vertical ("dopage" du serveur) à un système horizontal (ajout de machines "basiques")
- ▶ Contraignant avec des SGBD relationnels (*cf M2TI - GGMD*)

Nouveaux besoins :

- ▶ Passage à l'échelle (meilleures performances)
- ▶ Forte disponibilité, résistance aux pannes
- ▶ Gestion flexible des données (schémas dynamiques)

Motivation

Distribution des données sur différents serveurs et "data centers" :

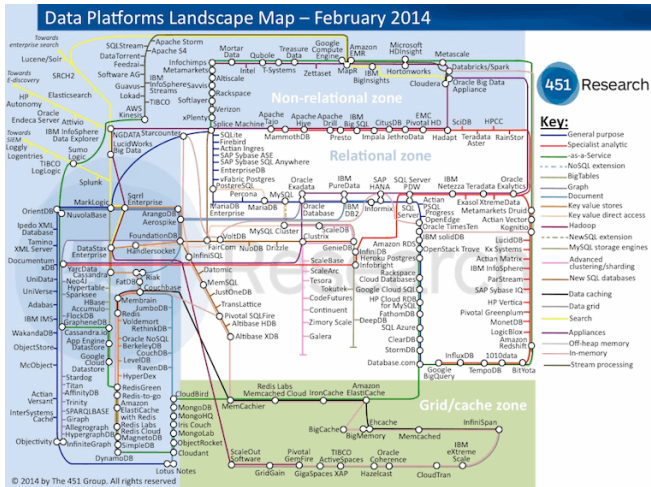
- ▶ Passage d'un système vertical ("dopage" du serveur) à un système horizontal (ajout de machines "basiques")
- ▶ Contraignant avec des SGBD relationnels (*cf M2TI - GGMD*)

Nouveaux besoins :

- ▶ Passage à l'échelle (meilleures performances)
- ▶ Forte disponibilité, résistance aux pannes
- ▶ Gestion flexible des données (schémas dynamiques)

Mouvement NoSQL, NotOnlySQL, Non Relationnel, NewSQL, ...

Panorama des SGBD (2016)



<https://451research.com/451-research-data-platform-map>

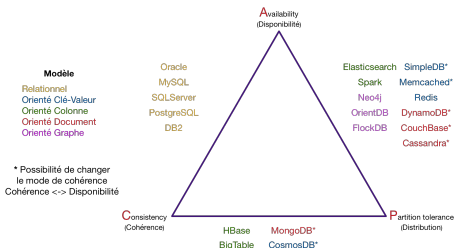
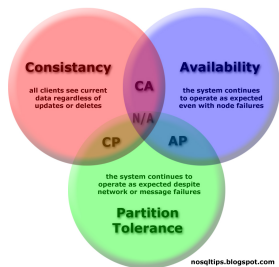
Caractéristiques des SGBD non-relationnels

SGBD non-relationnel \approx entrepôt clé-valeur fortement optimisé

- ▶ Modèle de données non relationnel (clé-valeur, colonne, document, graphe, etc.)
- ▶ Pas de jointure (données dénormalisées)
- ▶ Schéma optionnel (pas de contraintes sur les données, mais validation possible selon un schéma)
- ▶ Partitionnement horizontal des données, répliquées sur plusieurs machines
- ▶ Absence de garantie de cohérence (mais mécanismes pour l'atteindre à moyen terme)

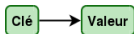
Théorème CAP (ou de Brewer)

Un système distribué ne peut garantir en même temps la cohérence, la disponibilité et la résistance au partitionnement

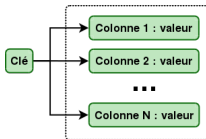


<http://www.julianbrowne.com/article/viewer/brewers-cap-theorem>
<http://openclassrooms.com/fr/courses/4462426>

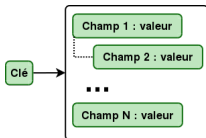
Modèles (principaux) des SGBD non-relationnels



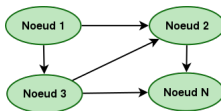
Modèle clé-valeur



Modèle colonne



Modèle document



Modèle graphe

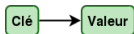


Leverage the NoSQL boom

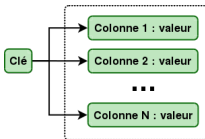
<http://db-engines.com/>

Davoudian et al. [A Survey on NoSQL Stores](#), CSUR (2018)

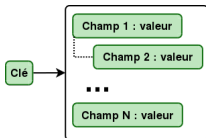
Modèles (principaux) des SGBD non-relationnels



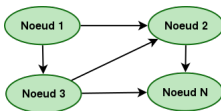
Modèle clé-valeur



Modèle colonne



Modèle document



Modèle graphe



Leverage the NoSQL boom

Dans la suite, les SGBD orientés document

<http://db-engines.com/>

Davoudian et al. [A Survey on NoSQL Stores](#), CSUR (2018)

Plan

Modèle de données orienté document

Caractéristiques de MongoDB

Requêtage avec MongoDB

Rappels

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

- ▶ Un mode de représentation des données, via le langage de définition de données (LDD)
- ▶ Un mode de représentation des contraintes sur ces données, via le LDD
- ▶ Un ensemble d'opérations (CRUD) pour manipuler les données, via le langage de manipulation de données (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

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>)* ] | []
```

Grammaire pour représenter un document. 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

Adapté de Li et al., [Design Issues of JPQ: a Pattern-based Query Language for Document Databases](#), CoRR (2015)

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"]
  }
}
```

Ce document est-il conforme à la grammaire?

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>)* ] | []
```

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

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

Grammaire pour la sélection

Soit une collection de documents \mathcal{C} , un langage de manipulation permet de sélectionner les documents de \mathcal{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 - exemple

```
{  
  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")  
  
z: {taille: 2}  
  
x: "val_x" ou z.zx: {=: "vzx"}
```

Grammaire pour la sélection - exemple

```
{  
  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 ?

<code>x: "vx"</code>	✓
<code>w: {>: 5}</code>	✗
<code>non(x: "vx")</code>	✗
<code>z: {taille: 2}</code>	✓
<code>x: "val_x" ou z.zx: {=: "vzx"}</code>	✓

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 z
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

En résumé

Plusieurs grammaires pour les SGBD orientés document :

- ▶ Pour représenter les documents
- ▶ Pour sélectionner des documents
- ▶ Pour projeter les propriétés des documents



Le chat, Geluck

Dans la suite :

- ▶ Caractéristiques d'un SGBD orienté document, MongoDB
- ▶ Des exemples concrets de requête basées sur ces grammaires

Plan

Modèle de données orienté document

Caractéristiques de MongoDB

Requêtage avec MongoDB

Généralités

- ▶ SGBD orienté documents
- ▶ Open-source
- ▶ Populaire (5^{ème} 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



<http://www.mongodb.com/>

<http://robomongo.org/download>

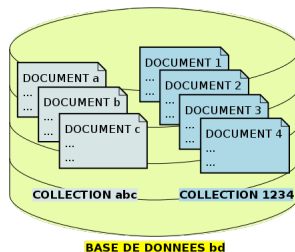
Concepts principaux - BD et collection

Base de données (\sim *base de données* en modèle relationnel) :

- ▶ Ensemble de collections
- ▶ Espace de stockage

Collection (\sim *table* en modèle relationnel) :

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



<http://docs.mongodb.com/manual/core/databases-and-collections/>

Concepts principaux - document

Document (*~ ligne, tuple* en modèle relationnel) :

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

BSON = "Binary JSON" (JavaScript Object) 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)

<http://docs.mongodb.com/manual/core/document/>
<http://bsonspec.org/>

Concepts principaux - document (2)

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",
    ...
  }
}
```

Concepts principaux - document (3)

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

← field: value
← field: value
← field: value
← field: value

Exemple de document au format BSON

Définition possible de contraintes sur les champs (validation)

<http://docs.mongodb.com/manual/introduction/>

<http://docs.mongodb.com/manual/core/document-validation/>

Relations inter-documents

Relation entre les documents de différentes collections :

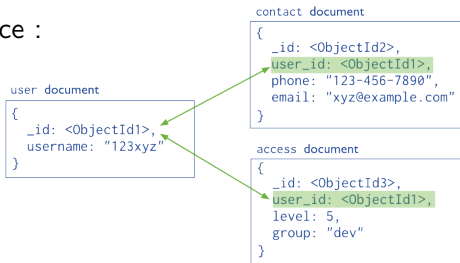
- ▶ Par référence : l'identifiant d'un document (son "_id") est utilisé comme valeur attributaire dans un autre document
 - ▶ se rapproche du modèle de données normalisées
 - ▶ nécessite des requêtes supplémentaires côté applicatif

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

<http://docs.mongodb.com/manual/core/data-modeling-introduction/>

Relations inter-documents (2)

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 \Rightarrow ajout d'un champ à tout moment
- ▶ Pas de jointure \Rightarrow redondances possibles
- ▶ Inclusion \Rightarrow moins de lectures
- ▶ Opérations atomiques sur un seul document, mais pas sur plusieurs \Rightarrow état incohérent temporaire (souvent tolérable)

<http://docs.mongodb.com/manual/core/data-model-design/>
<http://docs.mongodb.com/manual/core/schema-validation/>
<http://mongoosejs.com/>

Modélisation d'une BD

Considérations pour modéliser une BD au niveau physique :

- ▶ Accès par une clé (identifiant d'un document)
- ▶ Schéma optionnel \Rightarrow ajout d'un champ à tout moment
- ▶ Pas de jointure \Rightarrow redondances possibles
- ▶ Inclusion \Rightarrow moins de lectures
- ▶ Opérations atomiques sur un seul document, mais pas sur plusieurs \Rightarrow 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)

<http://docs.mongodb.com/manual/core/data-model-design/>

<http://docs.mongodb.com/manual/core/schema-validation/>

<http://mongoosejs.com/>

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 :

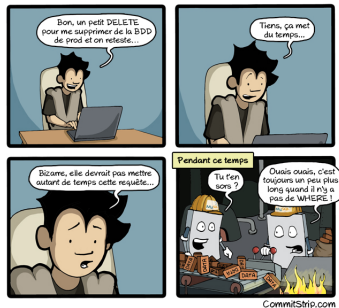
- ▶ éviter une forte redondance sans "gain" (livre/éditeur)
- ▶ des entités fortement connectées "*many to many*" (livres/auteurs)
- ▶ des données hiérarchiques (catégories/sous-catégories/...)

<http://docs.mongodb.com/manual/core/data-modeling-introduction/>

En résumé

MongoDB, un SGBD documents :

- ▶ Requêtes CRUD et requêtes agrégatives (regroupements)
- ▶ Indexation, administration
- ▶ Sharding (distribution des données) et réplication
- ▶ Map Reduce avec Javascript (distribution des traitements)
- ▶ Des "drivers" dans une quinzaine de langages (e.g., Jongo)



<http://docs.mongodb.org/ecosystem/drivers/>

<http://jongo.org/> ou <https://pymongo.readthedocs.io/>

Plan

Modèle de données orienté document

Caractéristiques de MongoDB

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

<https://www.mongodb.com/docs/manual/crud/>

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 **etu** contenant 4 documents représentant chacun un-e étudiant-e, son année d'inscription, ses groupes et ses notes

<https://pipoya.itch.io/pipoya-free-rpg-character-sprites-32x32>

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}`

<https://www.mongodb.com/docs/manual/reference/method/db.collection.find/>

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 (2)

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

annee : 2020 groupes : [A]	annee : 2022 groupes : [A]
annee : 2021 groupes : [A]	annee : 2020 groupes : [B]

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

Ana notes : [{note : 17}, {note : 18}] date_gen : '01/09' 	Bob notes : [{note : 19}, {note : 14}] date_gen : '01/09' 
Cya notes : [{note : 14}] date_gen : '01/09' 	Dan notes : [{note : 9}, {note : 16}] 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  
})
```

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

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

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]
```



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 }
  }
})
```

Ana

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

**Dan**

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



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

Ana

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

**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}]
```



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)

Interrogation - exemples avancés (2)

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 : 18),
 (ue : 'WEB', note : 14) }]

Cya

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

Dan

annee : 2022
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

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

curseur

`next()`

Ana

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



`next()`

Bob

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



...

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)
```

Ana

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



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


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}]
```



Cya

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



Bob

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



Zoé

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



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

Bob

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




Zoé

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




Cya

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



Ana

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



Dan

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



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

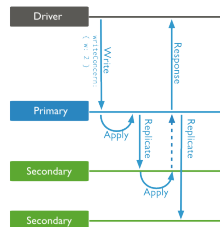
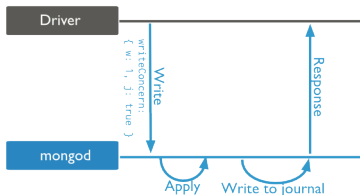
<https://www.mongodb.com/docs/manual/reference/method/db.collection.insertOne/>

<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"

<https://www.mongodb.com/docs/manual/reference/write-concern/>

Insertion - exemple

```
1 db.etu.insertOne({
2   _id: "Ela",
3   annee: 2022,
4   groupes: ["B", "B1"]
5 },
6 {
7   writeConcern: {
8     j: true
9   }
10 })
11
```

```
1 db.etu.insertMany([
2   {
3     _id: "Flo",
4     annee: 2022,
5     groupes: ["B", "B2"]
6   },
7   {
8     _id: "Gad",
9     annee: 2022
10  }
11 ])
```

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

```
1 db.etu.insertOne({
2   _id: "Ela",
3   annee: 2022,
4   groupes: ["B", "B1"]
5 },
6 {
7   writeConcern: {
8     j: true
9   }
10 })
11
```

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

```
1 db.etu.insertMany([
2   {
3     _id: "Flo",
4     annee: 2022,
5     groupes: ["B", "B2"]
6   },
7   {
8     _id: "Gad",
9     annee: 2022
10  }
11 ])
```

```
{
  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 *filter*)
 - ▶ *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 }}}  
)
```

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

Exemples d'une mise à jour de la note WEB de Cya (gauche) et d'une mise à jour de tous les documents sans note inférieure à 10 par l'ajout du nombre d'ECTS acquis (droite)

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  
}
```

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

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

Exemples d'une mise à jour de la note WEB de Cya (gauche) et d'une mise à jour de tous les documents sans note inférieure à 10 par l'ajout du nombre d'ECTS acquis (droite)

Suppression

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

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

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

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

Exemples de suppression du document d'identifiant Dan avec garantie d'écriture (gauche) et suppression des documents dont la date n'est pas 2022 (droite)

<https://www.mongodb.com/docs/manual/tutorial/remove-documents/>

Suppression

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

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

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

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

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

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

Exemples de suppression du document d'identifiant Dan avec garantie d'écriture (gauche) et suppression des documents dont la date n'est pas 2022 (droite)

<https://www.mongodb.com/docs/manual/tutorial/remove-documents/>

- ▶ **Big Data + web** \Rightarrow nouveaux besoins pour la modélisation, le stockage et le traitement de données **volumineuses**, véloces (**flux**) et variées (**hétérogénéité**)
- ▶ **Mouvement NoSQL / non-relationnel / NewSQL :**
 - ▶ paradigmes clé-valeur, colonne, document et graphe
 - ▶ théorème de CAP (consistency, availability, partition tolerance)
- ▶ **MongoDB**, un SGBD orienté document utilisant un langage de requêtage basé sur les motifs

Perspectives : traitement distribué (algèbre de collection, aggregation pipeline), répartition des données (M2TI - GGMD)