Introduction aux BD documents NoSql / mongodb

- Caractéristiques et limites de SQL
- Les BD Documents et NoSql : objectifs et principes
- Modélisation des données dans MogoDB
- Requêtes MongoDB
- L'interface php-mongo

SQL : caractéristiques

- Les bases de données relationnelles :
  - tables, attributs, ligne, clé primaire, clé étrangère
- Information **fortement structurée** et décomposée en unités élémentaires (entiers, chaines, dates…) regroupées en "tableaux"
- Les données complexes sont **décomposées** en tables multiples et doivent être **reconstruites** à l'aide d'opérations couteuses (jointure, sous-requêtes)
- Forte cohérence grâce aux transactions

SQL : les limites

- Limites sur la nature des données :
  - données peu structurées : textes, documentation ..
  - données dont la structure n'est pas figée
  - données agrégées complexes
- Limites sur le volume des données
  - Opérations couteuses (jointure) difficiles sur des grandes quantités de données
  - Le modèle de cohérence limite la possibilité de distribuer les données sur plusieurs serveurs pour améliorer les performances

Le théorème CDP

- Pour traiter des gros volumes, une solution consiste à **distribuer les données sur plusieurs serveurs** connectés entre eux.
- On est alors limité par le théorème CDP

  Il est impossible d'obtenir en même temps :
  - **Cohérence** des données : tous les serveurs disposent des mêmes données au même moment
  - **Disponibilité** : toute les requêtes obtiennent une réponse
  - **Partition** : on peut supporter une panne réseau partielle où certains serveurs sont isolés

Les sgbd SQL

- Schéma = structure des tables, figé et peu évolutif
- Privilégient **Cohérence** et **Disponibilité**
  - limitation du nombre de serveurs dans une base de données distribuée
- Peu adaptés :
  - aux bases de données de très grande taille nécessitant une forte distribution
  - aux données peu structurés ou très complexes

Les sgbd NoSQL

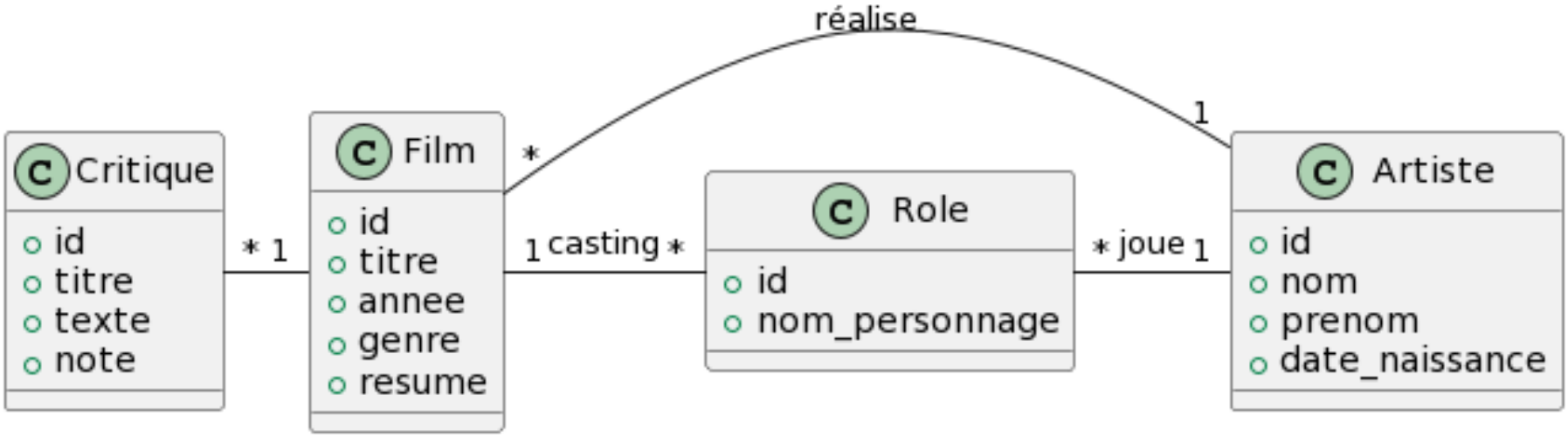
- Sgbd dont l'objectif est de dépasser les limites de SQL
- Traiter des volumes importants grâce à la distribution forte des données
  - Disponibilité et Partition au détriment de Cohérence
- Structure des données étendue ou assouplie
  - Cassandra : données = ensemble de couples clé-valeur avec un schéma défini et des types étendus
  - MongoDB : données = document json sans schéma établi

MongoDB : un sgbd *document*

- MongoDb est un sgbd NoSql dit "document"
- Une BD mongo est un ensemble de collections de **documents json** dont la structure est quelconque

SQL	mongo
Base de donnée	idem
table	Collection d'objets
ligne	objet json
Colonne (attribut)	propriété
schéma	-

modélisation



SQL :

```
film(#id, titre, année, genre, rea_id)
artiste(#id, nom, prenom, date_naiss)
role(#id, nom_personnage , a_id, f_id)
critique(#id, titre, texte, note, f_id)
```

Utilisation de la jointure

```
select titre, nom, prenom, nom_pers
from film, role, artiste
where titre = 'pulp fiction'
  and  film.id = role.f_id
  and role.a_id = artiste.id
```

modélisation dans mongodb

- les entités sont représentées par des objets JSON
  - pas de schéma prédéfini contraignant, la structure des entités n'est pas figée
- les associations peuvent être représentées
  - par référence vers un ou plusieurs objets, équivalent à une clé étrangère
  - par **imbrication d'objets**

Représentation avec imbrication des associations

```
{
  "_id" : 1,
  "titre" : "pulp fiction",
  "annee": 1994,
  "genre": "action",
  "directeur" : {
       "nom": "Tarantino", "prenom": "quentin", "date_naiss": 1963},
  "casting" : [
   {"perso": "vincent", "acteur": {"nom": "travolta", "date_n": 1954 }},
   {"perso": "butch",   "acteur": {"nom": "willis",  "date_n": 1955 }},
   {"perso": "jimmy",   "acteur": {"nom": "tarantino","date_n": 1963}}
  ]
  "critiques" : [
   {"titre": "bof", "texte": "ya mieux", "note": 3, "user_id": 3654de },
   {"titre": "super", "texte": "ya pas mieux",
             "note": 5, "user_id": 5487b },
   {"titre": "nul", "texte": "nul, nul", "note": 1, "user_id": e5643c },
  ]
}
```

associations sous forme de référence

sql vs. mongo

- associations 1--*
  - sql : clé étrangère côté *
  - mongo :
    - **imbrication** sous la forme d'un tableau d'objets
    - **référence** côté * et/ou tableau de références côté 1
- associations *--*
  - sql : table pivot avec clé étrangères, pas de données dupliquées
  - mongo :
    - **imbrication** sous la forme d'un tableau d'objets et duplication de données
    - tableau de références d'un côté ou des deux côtés

imbrication : intérêts

- Plus besoin de jointure !
- accès aux données agrégées en une seule requête
- Un document est autonome et auto-suffisant
- Plus adapté à la distribution des données : un document autosuffisant est plus facile à déplacer d'un serveur à un autre
- Mises à jour plus simple : 1 écriture suffit pour créer un document alors que + lignes de 3 tables en SQL

Oui, mais

- Hiérarchisation des accès : la représentation n'est pas symétrique. Dans le choix qui est fait, on privilégie les films
  - On ne peut pas accéder à un acteur en dehors des films dans lesquels il joue
- Perte d'autonomie des entités : un acteur n'existe pas en dehors des films – si on supprime 1 film, on risque de supprimer des acteurs
- Redondance de données

critères de choix

- Les données accédées simultanément doivent être stockées ensemble

- Privilégier **l'imbrication**
- en particulier lorsque une des entité a un rôle spécial dans l'association
  - l'association représente une agrégation
  - on accède toujours (ou souvent) à une entité et ses associées en même temps

critères de choix

- réserver les **références** aux cas :
  - les entités sont complètement indépendantes : créées, mises à jour, supprimées à des moments différents
  - la duplication de données est trop complexe à gérer, par exemple, mise à jour fréquente de données dupliquées,
  - document risquant de devenir très volumineux : cardinalité illimitée et/ou entités associées volumineuses
- Eviter de gérer des références bi-directionnelles couteuses à maintenir

Les données dans mongodb

- Une base de données mongodb est un ensemble de ***collections***
- Chaque collection est un ensemble de d'***objets json***
- Chaque objet json est identifié de manière unique par un identifiant, valeur de la propriété ***"_id"***

identifiants d'objets

- Tout objet dans 1 base mongo a un identifiant unique, valeur de la propriété **`"_id"`**
  - fourni par **l'application** lors de l'insertion
  - ou généré par le serveur sous la forme d'un **`ObjectId`**
- ObjectId : valeur 12 bytes ordonnée générée par le serveur
  - On peut obtenir la date de création
  - On peut convertir ObjectId ↔ String

les types

- format interne : BSON
- format externe : JSON étendu

string	"abcdefgh"
int32, double, Int64	1254, 4355665487, 345665.7675
decimal	{"$numberDecimal" : "10.02" }
Date après 1970	{"$date": "<ISO-8601 Date Format>" } {"$date": "2019-11-26" }
Date avant 1970	{"$date" : {"$numberLong" : "millis"} } {"$date" : {"$numberLong" : "-654455677"}
timestamp	{"$timestamp" : {"t" : "3344552627", "i":1}
ObjectId	{"$oid" : "5d505646cf6d4fe581014ab2"}
object	{ … }
array	[…]

Utiliser mongo

- mongoshell : shell javascript pour interroger interactivement un serveur mongod en cli
- Applications GUI : **compass**, NoSQLBooster, robomongo, mongo express
- Driver langages : python, java, node, php, go, ruby ...

connexion à une base mongo

mongoshell	mongodb/mongodb PHP library

$ mongo --host localhost:27017	`require_once "vendor/autoload.php" ;`
	
	`$c = new \MongoDB\Client("mongodb://localhost:27017");`

> use moviedb >	`$db = $c->moviedb;`

requêtes sur une base mongo

- Voir https://docs.mongodb.com/manual/crud/
- **Insertion de documents** :

```
db.collection.insertOne(
   <document>
)
db.collection.insertMany(
   <array of documents>
)
```

  - Insertion atomique
  - Si la collection n'existe pas elle est créée
  - Si _id est absent dans le document, un ObjectId est généré par le serveur

mongoshell

```
>db.movies.insertOne(  {
    "titre" : "joker",
   "annee": 2019,
   "genre": "marvel",
   "directeur" : { "nom": "Phillips", "prenom": "Todd"
   }
})

{ "acknowledged" : true,
   "insertedId" : ObjectId("5db96ead91c163ef3b02b68b")
}
```

php

```
$i=$db->movies->insertOne( [
    "titre"=>"joker",
    "annee"=>2019,
    "genre"=>"marvel",
    "directeur"=> ["nom"=>"Phillips", "prenom"=>"Todd"]
]);

print $i->getInsertedId();
```

Recherche de documents

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

```
db.movies.find(
     { annee: {$lt: 2000}  ,
       genre: "action" }  ,
     { titre: 1, annee: 1, acteurs:0 }
).sort( {titre : -1} )
```

collection

critère de recherche

projection

```
select titre, annee
 from movies
     where annee < 2000  ,
       and genre = "action"
 Order by titre desc
```

projection

```
{ titre: 1,
  annee: 1 }

{ acteurs: 0 }
```

Uniquement titre et annee

Tout sauf acteurs

Critères de recherche

```
{ field: value }

{ field: {$op : value} }
```

```
genre =  value

field op  value
```

- opérateurs de comparaison :

  `$eq, $ne, $gt, $lt, $gte, $lte, $in, $nin`

- Condition sur document imbriqué :

  ```
  { directeur.nom: "tarantino }
  ```

- Condition sur 1 élément de tableau imbriqué :

  ```
  { acteurs.datenaiss: {$gt 1960}}
  ```

exemples

mongoshell	php	SQL
`db.movies.find()`	`$db→movies→find( [] ) ;`	`Select * from movies`

`db.movies.find(`
`  { titre :"joker"}`
`)`
	`$db→movies`
	`  →find( [`
	`  "titre" => "joker"]) ;`
		`Select *`
		`  From movies`
		`  Where titre= 'joker'`

`db.movies.find(`
`  { },`
`  {titre:1, genre:1}`
`)`
	`$db→movies`
	`  →find( [],`
	`  ['projection'=>`
	`  [ "titre"=>1,`
	`  "genre"=>1 ]`
	`) ;`
		`Select titre, genre`
		`  From movies`

`db.movies.find({`
`  genre:{ $in:[`
`  "action",`
`  "thriller" ]`
`  )`
	`$db→movies→find(`
	`  [ "genre" =>`
	`  ['$in' => [`
	`  "action",`
	`  "thriller ]`
	`  ] ) ;`
		`Select *`
		`  from movies`
		`  where genre in (`
		`  'action', 'thriller')`

- supprimer des documents :

```
// uniquement le 1er sélectionné
db.movies.deleteOne(
     { annee: {$lt: 2000}  }
)

//toute la sélection
db.movies.deleteMany(
     { genre: "romantique"  }
)
```

Modifier des documents existants

```
db.collection.update(
   <query>,
   <update>,
   <options> )
```

```
// uniquement le 1er sélectionné
db.movies.update(
     { titre : "joker"  },

     { $set : {"genre" :"action"} }
)

//toute la sélection
db.movies.update(
     { genre: "action"  },
     { $set : { age : "13+"} },
     { multi : 1 }
)
```

collection

critère de sélection

modification

- opérateurs de modification :

  `$set, $inc, $mul, $rename, $unset, $currentDate`

- Remplacer un document existant : le document sélectionné est remplacé par le nouveau sans modification de son _id

```
db.collection.replaceOne(
   <filter>,
   <replacement>
)
```

```
db.movies.replaceOne(
  { titre : "joker"  },
  {
    titre : "le jocker",
    annee : 2002,
    genre : "anime"
  }
)
```

collection

critère de sélection

Nouveau doc.

- Index
  - Par défaut, un index sur _id
  - On peut créer un index sur un ou plusieurs champs
    - Ascendant : 1
    - Descendant : -1

```
db.movie.createIndex( { annee : 1 } )
```