TP final : Création d'application pour le Cloud

Yasmeen TRifiss - KilLian VENEZIA - Tara Zhong

Développement d'application pour le Cloud | 29-12-2017

Table des matières

[Introduction 3](#_Toc504221048)

[1.1 Jeux de données 4](#_Toc504221049)

[1.2 Spécification des besoins 4](#_Toc504221050)

[1.3 Dénormalisation 4](#_Toc504221051)

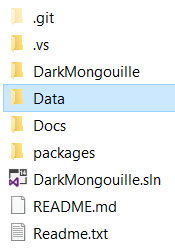
[1.3.1 Schéma 4](#_Toc504221052)

[1.3.1 Transformation 6](#_Toc504221053)

# **Introduction**

Ce document est le rapport pour notre application finale de création d’application pour le Cloud.

Vous trouverez dans notre dossier aussi disponible sur <https://github.com/yasmeentrifiss/DarkMongouille>

* Le code source de l’application est visible en ouvrant le .sln à l’aide de visual studio.
* Le readme est disponible en .txt ou bien sur le répertoire github.
* Dans le dossier Data :
* Vous trouverez :
  + Jeu de données original
  +  Jeu de données transformé

# **1.1 Jeux de données**

Pour réaliser cette application, nous utiliserons la base de données Sakila : <https://relational.fit.cvut.cz/dataset/Sakila>. Cette base de données est une base de données fictive qui simule la gestion d’une cinémathèque (lieu où l’on peut louer des DVDs)

# 1.2 Spécification des besoins

Pour pouvoir orienter les choix des étapes suivantes, il va falloir étudier les besoins de l’application :

**— Définir, en langage courant, 4 types d’interrogations sur votre jeu de données. On estimera que celles-ci sont effectuées très fréquemment. Positionnez-vous comme un utilisateur standard de l’application.**

* Quels sont les titres de films dont la classification cinématographique est "..."

Les valeurs sous classifications Américaines sont G, PG, PG-13, R, ou NC-17.

Ex :

- G (Audience générale): Pas de sexe ou nudité, abus de substances ou violence.

- PG (Sous surveillance parentale): Certaines scènes peuvent ne pas convenir aux enfants (violence, langage vulgaires mais pas de violence physique ou usages de drogues)

- …

Source : https://www.thoughtco.com/how-does-a-movie-get-its-rating-2423408

* Quels sont les informations du film dont le titre est « »
* Quels sont les films dont le genre est « »
* Quels sont les films dont le prénom de l’acteur/actrice est « »

BONUS

* Donner la liste de tous les films

**— Définir, en langage courant, 2 types d’interrogations lourdes sur votre jeu de données. Ces requêtes seront effectuées moins fréquemment. Positionnez-vous comme un analyste ou un décisionnaire de l’application.**

* Afficher pour un id de film donné, chaque inventaire pour ce film et la liste des locations
* Pour chaque catégorie donner le nombre de notes et la moyenne des notes

BONUS

* Afficher pour chaque inventaire la liste des locations

— Définir des indicateurs de statistiques pertinents sur votre jeu de données. Positionnez-vous comme administrateur de la base de données.

Les indicateurs sont inventory, rental, client.

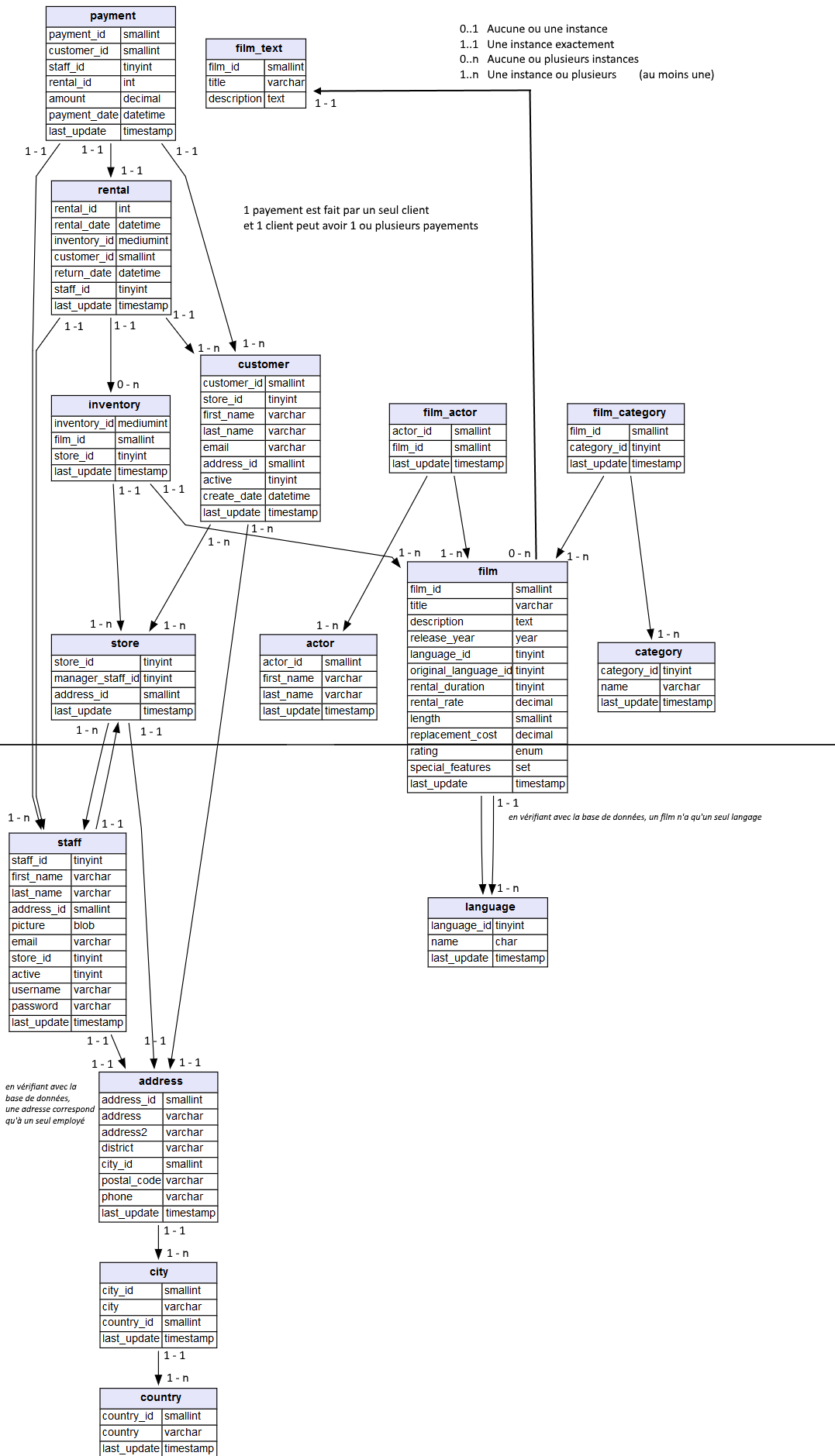
Pour le business user, il est important de gérer les inventaires et de pouvoir observer les locations effectuées sur chaque produit. Normalement la base de données de films est censée être plus ou moins fixe, par contre le nombre de location peut s’accroître rapidement.

# 1.3 Dénormalisation

## 1.3.1 Schéma

Afin de réaliser la dénormalisation, nous avons définis les cardinalités de chaque table pour regrouper entre elles les plus pertinentes en observant les données dans la base de données

**Schéma relationnel initial Sakila**



Nous avons supprimé les redondances rencontrées (par ex la table film\_text comporte les mêmes informations que dans la table film (title et description)).

Les tables finales sont :

**Film**: qui contient la table language convertie en un seul string (dans la base de données on a observé qu’une seule unique langue), la table actor convertie en liste des acteurs, et la table category convertie en liste des catégories en plus. (cf structure json)

**Customer** : qui contient la table adresse du customer convertie en attributs (address, district,city, postal\_code, country et phone), les attributs « city » de la table city et « country » de la table country ont directement été placé à côté de l’adresse. (cf structure json)

**Rental** : qui contient la table payment en plus convertie en attributs. (cf structure json)

**Store** : qui contient la table adresse du store convertie en attributs (address, district,city, postal\_code, country et phone), les attributs « city » de la table city et « country » de la table country ont directement été placé à côté de l’adresse. (cf structure json)

**Staff** : qui contient la table adresse du staff convertie en attributs (address, district,city, postal\_code, country et phone), les attributs « city » de la table city et « country » de la table country ont directement été placé à côté de l’adresse. (cf structure json)

**Inventory** : aucune modification sur cette table. (cf structure json)

## 1.3.2 Transformation

Pour la transformation nous avons stocké les données dans une base de données relationnelle. Les documents destinations sous format csv ont été produits à l’aide de requêtes SQL simples.

Puis nous avons importé les collections et converti les fichiers csv en fichiers json sous MongoDB pour les tables suivantes : customer, rental, store, staff, inventory. Une fois en csv nous pouvons directement formatter les fichiers en les important depuis mongoDb pour obtenir les collections json.

Pour la table « film », les jointures SQL ne nous permettait pas de créer une liste de tous les acteurs et une liste des catégories dans la table film.

Nous avons donc importé les 5 tables film, actor, film\_actor, category, film\_category pour faire des jointures à l’aide de $Lookup.

**ETAPES :**

### 1ere jointure

On fait tout d’abord une jointure avec la table film\_actor et actor en rajoutant les données de la table actor dans film\_actor (avec mongodb 3.6)

db.film\_actor.aggregate**([**

**{**

$lookup**:**

**{**

from**:** "actor"**,**

localField**:** "actor\_id"**,**

foreignField**:** "actor\_id"**,**

as**:** "actors\_data"

**}**

**},**

**{**

$replaceRoot**:** **{** newRoot**:** **{** $mergeObjects**:** **[** **{** $arrayElemAt**:** **[** "$actors\_data"**,** 0 **]** **},** "$$ROOT" **]** **}** **}**

**},**

**{** $project**:** **{** actors\_data**:** 0, \_id :0**}** **}**

**])**

De même pour la jointure avec category et film\_category

db.film\_category.aggregate**([**

**{**

$lookup**:**

**{**

from**:** "category"**,**

localField**:** "category\_id"**,**

foreignField**:** "category\_id"**,**

as**:** "category\_data"

**}**

**},**

**{**

$replaceRoot**:** **{** newRoot**:** **{** $mergeObjects**:** **[** **{** $arrayElemAt**:** **[** "$category\_data"**,** 0 **]** **},** "$$ROOT" **]** **}** **}**

**},**

**{** $project**:** **{** category\_data**:** 0, \_id :0**}** **}**

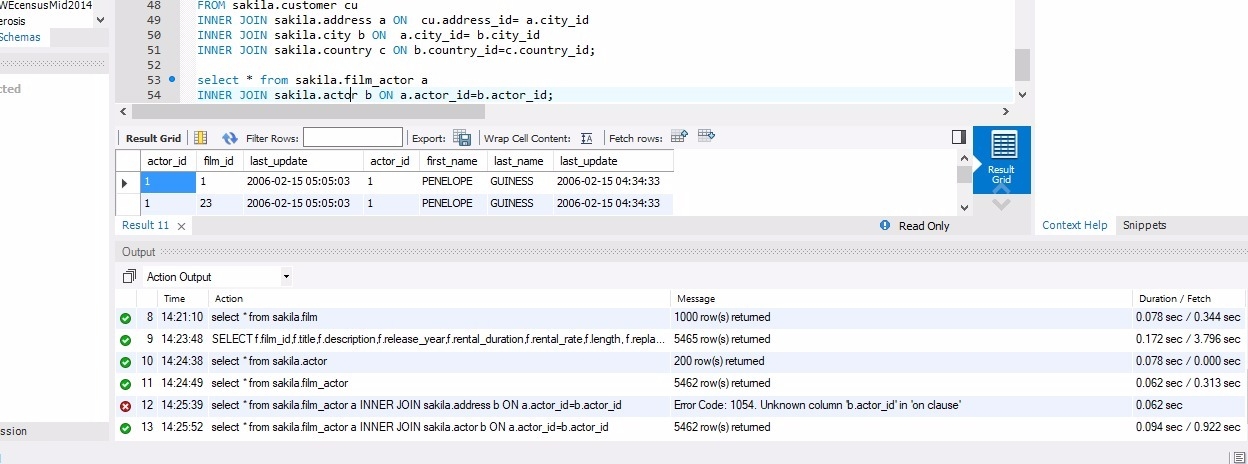
**])**

*~~Afficher les 5000 résultats de actor\_film, le document est limité à une taille de 1000~~*

[**~~https://docs.mongodb.com/manual/tutorial/iterate-a-cursor/~~**](https://docs.mongodb.com/manual/tutorial/iterate-a-cursor/)

***N’ayant pas réussis à avoir + de 1000documents en résultats sous Studio 3T nous avons décidé de faire la jointure en SQL***

Ex de requête pour la jointure actor et film\_actor



**Nous avons ensuite exporté les résultats sous une autre collection appelée actor\_film et category\_film**

### 2e jointure

Après avoir importé la table film originale sous le nom de film\_genuine, nous avons rassemblé les 2 jointures avec les 2 tables créées.

db.film\_genuine.aggregate**([**

**{**

$lookup**:**

**{**

from**:** "actor\_film"**,**

localField**:** "film\_id"**,**

foreignField**:** "film\_id"**,**

as**:** "actors"

**}**

**},**

**{**

$lookup**:**

**{**

from**:** "category\_film"**,**

localField**:** "film\_id"**,**

foreignField**:** "film\_id"**,**

as**:** "category"

**}**

**}**

**])**

On exporte le tout dans une collection appelé film

**Puis supprimer les \_ids générés par actor\_film et category\_film**

**(https://www.mkyong.com/mongodb/mongodb-how-to-remove-a-field-from-document/)**

db.film.find**({})**.forEach**(**function**(**doc**)** **{**

var actors = doc.actors**;**

for**(**var i = 0**;** i < actors.length**;** ++i**)** **{**

var x = actors**[**i**];**

delete **(**x**[**"\_id"**]);**

**}**

db.film.save**(**doc**);**

**});**

db.film.find**({})**.forEach**(**function**(**doc**)** **{**

var category = doc.category**;**

for**(**var i = 0**;** i < category.length**;** ++i**)** **{**

var x = category**[**i**];**

delete **(**x**[**"\_id"**]);**

**}**

db.film.save**(**doc**);**

**});**

db.film.find**();**

### Résultats

**Après transformation voici nos structures pour chaque table** :

6 tables: film, customer, rental, store, staff, inventory

Exemple d’objet de la table **film**

**{**

"\_id" **:** ObjectId**(**"5a63465df831b92d8cffbbc7"**),**

"film\_id" **:** NumberInt**(**2**),**

"title" **:** "ACE GOLDFINGER"**,**

"description" **:** "A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China"**,**

"release\_year" **:** NumberInt**(**2006**),**

"language\_id" **:** NumberInt**(**1**),**

"original\_language\_id" **:** "NULL"**,**

"rental\_duration" **:** NumberInt**(**3**),**

"rental\_rate" **:** 4.99**,**

"length" **:** NumberInt**(**48**),**

"replacement\_cost" **:** 12.99**,**

"rating" **:** "G"**,**

"special\_features" **:** "Trailers,Deleted Scenes"**,**

"last\_update" **:** "2006-02-15 05:03:42"**,**

"actors" **:** **[**

**{**

"\_id" **:** ObjectId**(**"5a634724f831b92d8cffc183"**),**

"actor\_id" **:** NumberInt**(**19**),**

"film\_id" **:** NumberInt**(**2**),**

"first\_name" **:** "BOB"**,**

"last\_name" **:** "FAWCETT"**,**

"last\_update" **:** ISODate**(**"2006-02-15T04:34:00.000+0000"**)**

**},**

**{**

"\_id" **:** ObjectId**(**"5a634724f831b92d8cffc887"**),**

"actor\_id" **:** NumberInt**(**85**),**

"film\_id" **:** NumberInt**(**2**),**

"first\_name" **:** "MINNIE"**,**

"last\_name" **:** "ZELLWEGER"**,**

"last\_update" **:** ISODate**(**"2006-02-15T04:34:00.000+0000"**)**

**},**

**{**

"\_id" **:** ObjectId**(**"5a634724f831b92d8cffc902"**),**

"actor\_id" **:** NumberInt**(**90**),**

"film\_id" **:** NumberInt**(**2**),**

"first\_name" **:** "SEAN"**,**

"last\_name" **:** "GUINESS"**,**

"last\_update" **:** ISODate**(**"2006-02-15T04:34:00.000+0000"**)**

**},**

**{**

"\_id" **:** ObjectId**(**"5a634725f831b92d8cffd0cb"**),**

"actor\_id" **:** NumberInt**(**160**),**

"film\_id" **:** NumberInt**(**2**),**

"first\_name" **:** "CHRIS"**,**

"last\_name" **:** "DEPP"**,**

"last\_update" **:** ISODate**(**"2006-02-15T04:34:00.000+0000"**)**

**}**

**],**

"category" **:** **[**

**{**

"\_id" **:** ObjectId**(**"5a634794f831b92d8cffd78c"**),**

"film\_id" **:** NumberInt**(**2**),**

"category\_id" **:** NumberInt**(**11**),**

"name" **:** "Horror"**,**

"last\_update" **:** ISODate**(**"2006-02-15T04:46:00.000+0000"**)**

**}**

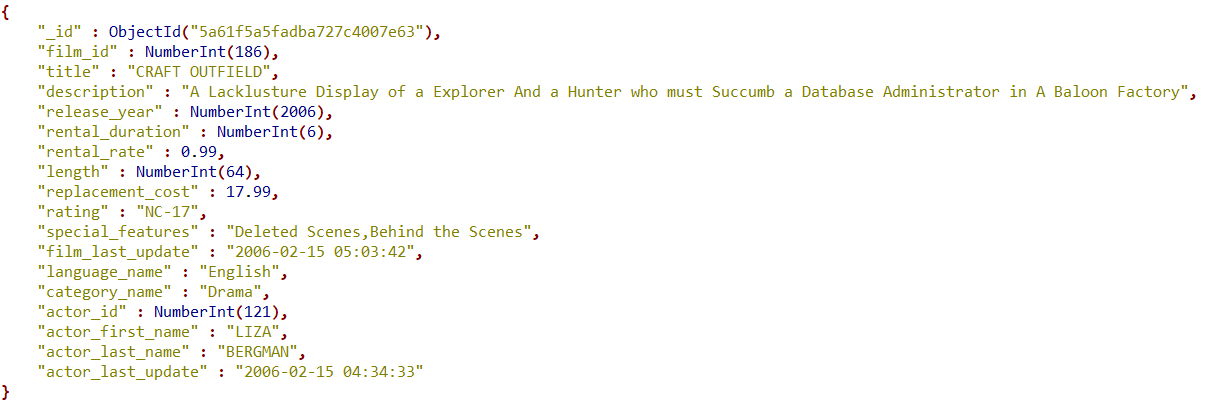
**]**

**}**

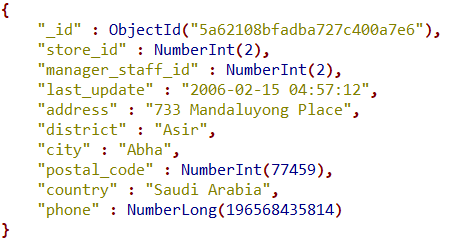
Exemple d’objet de la table **customer**



Exemple d’objet de la table **rental**



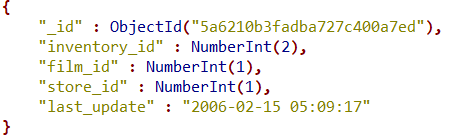
Exemple d’objet de la table **store**



Exemple d’objet de la table **staff**



Exemple d’objet de la table **inventory**



# 1.4 Vues

“Vous présenterez les choix d’implémentation effectués pour chaque vue., Expliquer les différentes vues réalisées et les choix effectués, Donner les requêtes MongoDB intégrées à chacunes des vues »

## 1.4.1 Utilisateurs standard

*Fournir à un utilisateur standard un accès aux résultats des 4 requêtes proposées. La vue n’a pas forcément besoin de fournir la requête mais seulement une vision sur les données interrogées.*

Toutes les requêtes sont présentes dans la classe Connection

* Quels sont les titres de films dont la classification cinématographique est « »

public void RateRequest(string rate)

(Equivalent sous mongoDb de db.film.find**({**"rating"**:**"PG"**}) )**

**Pour savoir la liste des classification possible, nous avons fait une requête simple sous mongoDb :** db.film.distinct("rating")

* Quels sont les informations du film dont le titre est « »

public void TitleRequest(string title)

* Quels sont les films dont le genre est « »

public void CategoryRequest(string category)

**Pour savoir la liste des catégories possible, nous avons fait une requête simple sous mongoDb :** db.film.distinct("category.name")

* Quels sont les films dont le prénom de l’acteur/actrice est « »

public void ActorRequest(string fname)

* Donner la liste de tous les films disponibles.

public void DisplayAllFilms()

## 1.4.2 Analyste/Décisionnaire

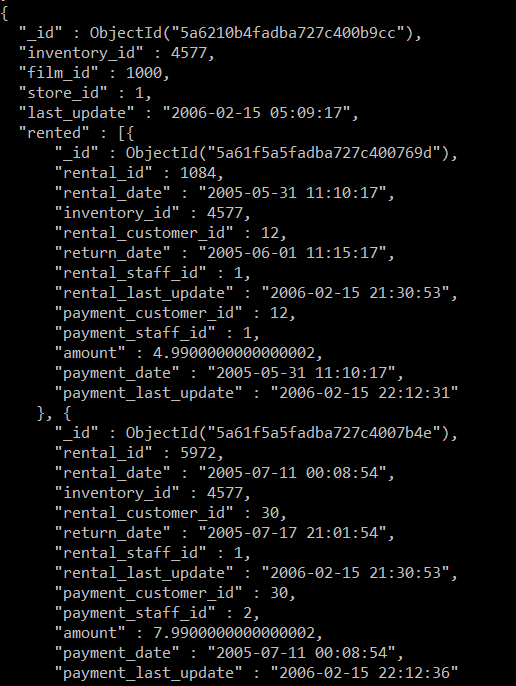
*Fournir à votre Data Analyst ou votre Business User une vue permettant de fournir les indicateurs proposés.*

*L’interface devra permettre de paramétrer les requêtes avec des valeurs (soit à la main, soit par menu déroulant).*

* Afficher, pour un id de film donné, chaque inventaire pour ce film et la liste des locations

public void InventoryRequestById(string filmId)

ex de résultat :



Equivalent de la requête mongodb suivante :

db.inventory.aggregate**([**

**{**

$lookup**:{**

from**:** "rental"**,**

localField**:** "inventory\_id"**,**

foreignField**:** "inventory\_id"**,**

as**:** "rented"

**}**

**},**

**{**$match**:{**film\_id **:**1**}}**

**])**

* Donner pour chaque inventaire la liste des locations

public void InventoryRequest()

* Pour chaque catégorie donner le nombre de notes et la moyenne des notes (utilise mapReduce)

public void CategoryRatingRequest()

## 1.4.3 Administrateur

*L’administrateur de la base MongoDB doit pouvoir récupérer différentes statistiques pour pouvoir faire évoluer le cluster en fonction de la charge. La vue doit pouvoir fournir les informations suivantes :*

*— Les statistiques sur les données définies dans la spécification des besoins*

*— Consulter l’état du cluster : nombre de shards, nombre de réplicats par shard*

*— Répartition des données sur les shards (nombre de documents)*

*— Indexes existants sur les données*

*Afin de créer les replicats set nous avons suivis les instructions :* [*http://chewbii.com/tp-mongodb-replication-et-sharding/*](http://chewbii.com/tp-mongodb-replication-et-sharding/)

1. **Dans le dossier data (ex : C:\data)**

**Créer chaque répertoire correspondant aux réplicas set**

Ex : Dans C:\data

1. **Pour chaque serveur ouvrir une console :**

Dans le repertoire de mongodb ex : C:\Program Files\MongoDB\Server\3.6\bin)

Ouvrir une fenêtre de commande (Taper cmd.exe dans le chemin)

1er serveur :taper mongod -replSet RS1 -dbpath C:\data\db -port 27017



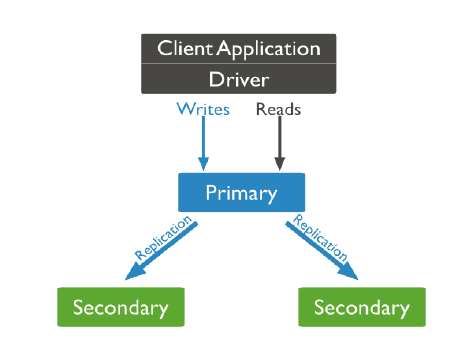
2e serveur : replicat Set : taper mongod -replSet RS1 -dbpath C:\data\rs1 -port 27018



3e serveur : replicat Set : taper mongod -replSet RS1 -dbpath C:\data\rs2 -port 27019



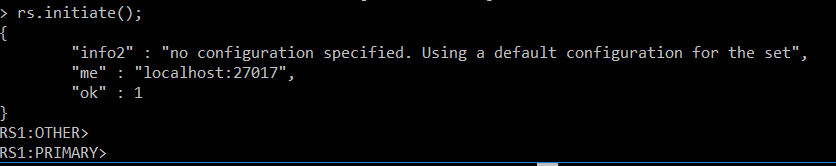
4e serveur : replicat Set : taper mongod -replSet RS1 -dbpath C:\data\rs4 -port 27020



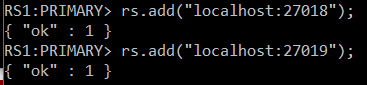
1. **Ouvrir une nouvelle console pour se connecter au serveur qui sera choisi comme serveur primaire :**

mongo -port 27017

Choix du serveur primaire :



Ajout de chaque réplicat :



Résultat de la configuration :

rs.conf();

{

"\_id" : "RS1",

"version" : 3,

"protocolVersion" : NumberLong(1),

"members" : [

{

"\_id" : 0,

"host" : "localhost:27017",

"arbiterOnly" : false,

"buildIndexes" : true,

"hidden" : false,

"priority" : 1,

"tags" : {

},

"slaveDelay" : NumberLong(0),

"votes" : 1

},

{

"\_id" : 1,

"host" : "localhost:27018",

"arbiterOnly" : false,

"buildIndexes" : true,

"hidden" : false,

"priority" : 1,

"tags" : {

},

"slaveDelay" : NumberLong(0),

"votes" : 1

},

{

"\_id" : 2,

"host" : "localhost:27019",

"arbiterOnly" : false,

"buildIndexes" : true,

"hidden" : false,

"priority" : 1,

"tags" : {

},

"slaveDelay" : NumberLong(0),

"votes" : 1

}

],

"settings" : {

"chainingAllowed" : true,

"heartbeatIntervalMillis" : 2000,

"heartbeatTimeoutSecs" : 10,

"electionTimeoutMillis" : 10000,

"catchUpTimeoutMillis" : -1,

"catchUpTakeoverDelayMillis" : 30000,

"getLastErrorModes" : {

},

"getLastErrorDefaults" : {

"w" : 1,

"wtimeout" : 0

},

"replicaSetId" : ObjectId("5a6496f69f69aa6764420c62")

}

}

Création du serveur Arbitre :



Ajout Arbitre :



**TEST de réplication avec la base de donnée sakila**



Requêtes simples avec replicat 1 et 2







**Installation du sharding**

**Creation du routeur de sharding**

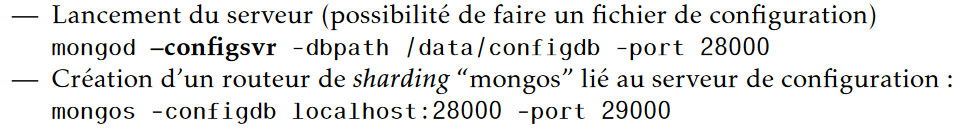


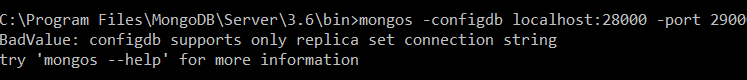
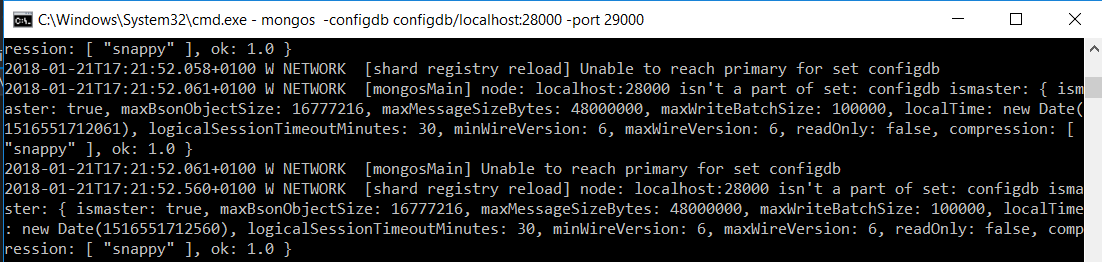
**Taper la commande :** mongos --configdb configdb/localhost:28000 --port 29000

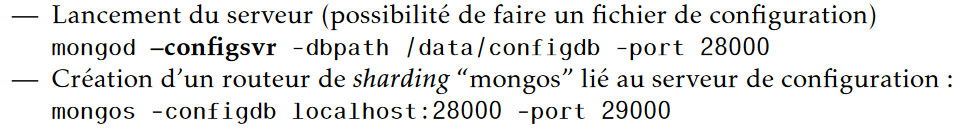
Lancement de 2 serveurs de données :

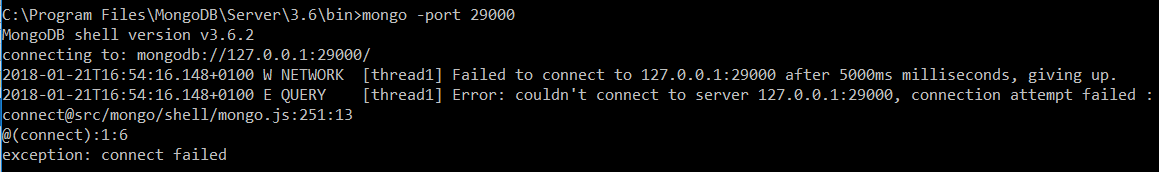
>mongod -dbpath C:/data/sh1 -port 27017

>mongod -dbpath C:/data/sh2 -port 27018



1e Erreur lors lancement du routeur de sharding :  2e Erreur lors lancement du routeur de sharding : 



mongo -port 29000 

Résultat du status dans status.txt

N’ayant pas réussis à créer les clusters nous n’avons pas implémenté :

— Les statistiques sur les données définies dans la spécification des besoins

— Consulter l’état du cluster : nombre de shards, nombre de réplicats par shard

— Répartition des données sur les shards (nombre de documents)

— Indexes existants sur les données

Il aurait fallu coder l’équivalent de rs.status() ; (mais nous n’avons ni la gestion des shards ni de réplicats)

~~Il aurait fallu modifier l’appel de la base de donnée :~~

~~MongoClient mongoClient = new MongoClient(Arrays.asList(~~

~~new ServerAddress("localhost", 27017),~~

~~new ServerAddress("localhost", 27018),~~

~~new ServerAddress("localhost", 27019)));~~

# Application

Les deux parties sont les suivantes :

— La connexion à MongoDB

— Chaque vue

# References

MongoDB C# Driver Cheat Sheet : <http://www.layerworks.com/blog/2014/11/11/mongodb-shell-csharp-driver-comparison-cheat-cheet>

Aggregation in c# :<https://www.chrismckee.co.uk/mongo-aggregation-in-c/>

How to MongoDB in C# : <https://blog.oz-code.com/how-to-mongodb-in-c-part-1/>

Mapping Classes : <http://mongodb.github.io/mongo-csharp-driver/2.0/reference/bson/mapping/>

Map reduce example : <https://gist.github.com/maxwedwards/4588041>

MongoDB Driver Admin Quick Tour : <http://mongodb.github.io/mongo-csharp-driver/2.4/getting_started/admin_quick_tour/>

Database Commands : <https://docs.mongodb.com/manual/reference/command/>