TP final : Création d'application pour le Cloud

Yasmeen TRifiss - KilLian VENEZIA - Tara Zhong

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

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 note est supérieure à « » et qui sont disponibles
* Quels sont les titres de films dont le genre est « » et qui sont disponibles
* Quels sont les titres de films dont l'acteur est « » et qui sont disponibles
* Donner la liste de tous les films disponibles.

*Pour savoir si un DVD est disponible, il faut regarder la date de retour «*return\_date » *dans la table rental et regarder si la date de retour «*rental\_duration » dans la table film *par rapport à la date de l’emprunt «*rental\_date »

— Définir, en langage courant, 2 tymopes 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.

* Trier la liste de films les plus loués (avec le nombre de fois) par ordre décroissant
* Pour chaque genre donner le nombre de notes et la moyenne des notes par ordre décroissant.

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

Les indicateurs sont les catégories, acteurs, clients.

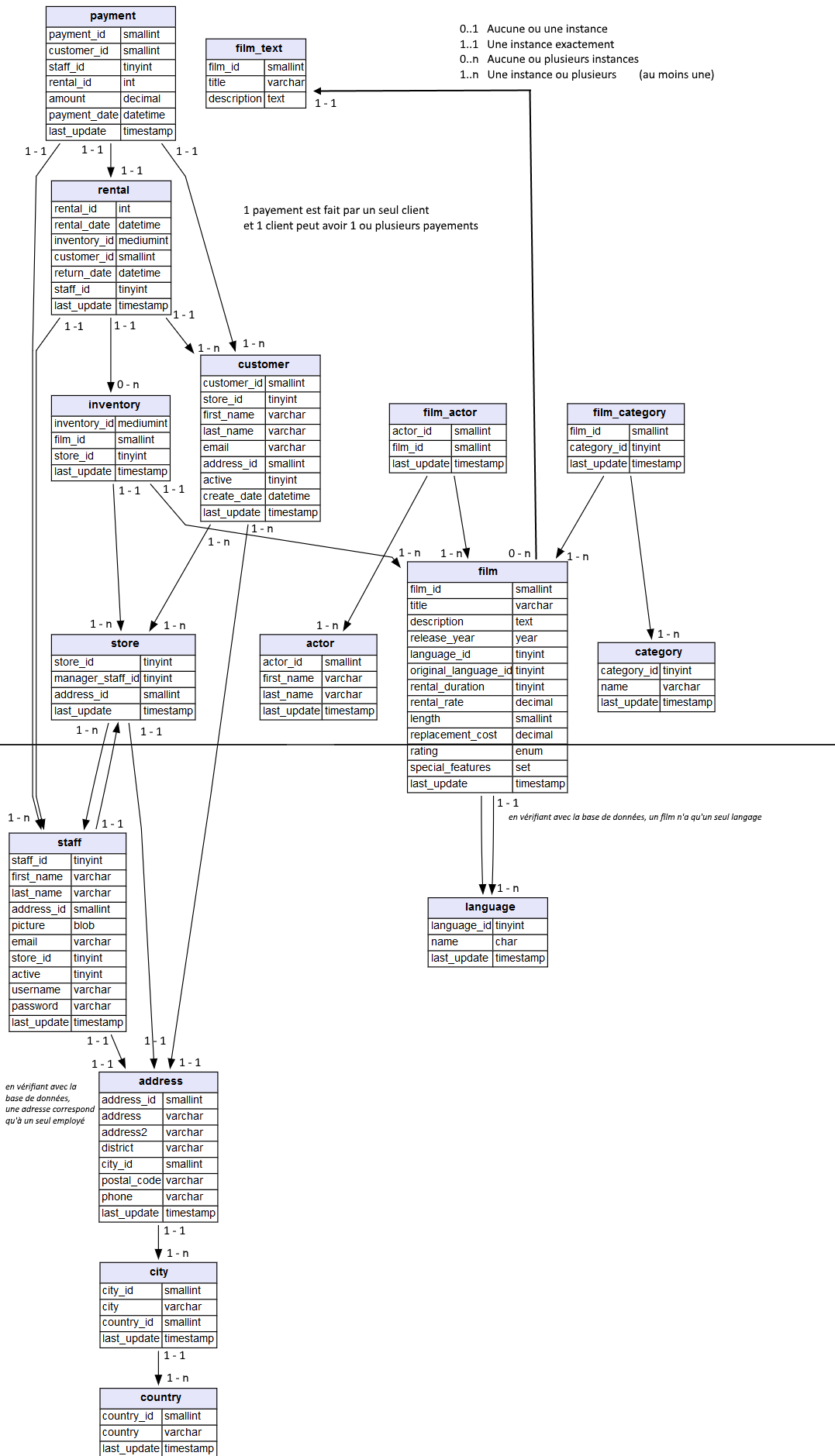
Ex de requêtes : Quels sont les clients qui n’ont retourné leur DVD.

# 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)

## 1.3.1 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 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 :**

On fait tout d’abord une jointure avec la table film\_actor et actor à gauche en rajoutant les données de la table actor dans film\_actor

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**}** **}**

**])**

Nous avons ensuite exporté le résultats sous une autre collection appellée actor\_film et category\_film

Puis fait 2 jointures avec les 2 tables créées.

db.film.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**();**

**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/)

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

Exemple d’objet de la table **film**

**{**

"\_id" **:** ObjectId**(**"5a62828d3c53ab26a4a2565d"**),**

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

"title" **:** "ACADEMY DINOSAUR"**,**

"description" **:** "A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies"**,**

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

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

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

"rental\_duration" **:** NumberInt**(**6**),**

"rental\_rate" **:** 0.99**,**

"length" **:** NumberInt**(**86**),**

"replacement\_cost" **:** 20.99**,**

"rating" **:** "PG"**,**

"special\_features" **:** "Deleted Scenes,Behind the Scenes"**,**

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

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

**{**

"\_id" **:** ObjectId**(**"5a62839e3c53ab26a4a25f0c"**),**

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

"first\_name" **:** "PENELOPE"**,**

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

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

"film\_id" **:** NumberInt**(**1**)**

**},**

**{**

"\_id" **:** ObjectId**(**"5a62839e3c53ab26a4a25fe0"**),**

"actor\_id" **:** NumberInt**(**10**),**

"first\_name" **:** "CHRISTIAN"**,**

"last\_name" **:** "GABLE"**,**

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

"film\_id" **:** NumberInt**(**1**)**

**},**

**{**

"\_id" **:** ObjectId**(**"5a62839e3c53ab26a4a260f5"**),**

"actor\_id" **:** NumberInt**(**20**),**

"first\_name" **:** "LUCILLE"**,**

"last\_name" **:** "TRACY"**,**

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

"film\_id" **:** NumberInt**(**1**)**

**},**

**{**

"\_id" **:** ObjectId**(**"5a62839e3c53ab26a4a26218"**),**

"actor\_id" **:** NumberInt**(**30**),**

"first\_name" **:** "SANDRA"**,**

"last\_name" **:** "PECK"**,**

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

"film\_id" **:** NumberInt**(**1**)**

**}**

**],**

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

**{**

"\_id" **:** ObjectId**(**"5a6281423c53ab26a4a23d0b"**),**

"category\_id" **:** NumberInt**(**6**),**

"name" **:** "Documentary"**,**

"last\_update" **:** "2006-02-15 05:07:09"**,**

"film\_id" **:** NumberInt**(**1**)**

**}**

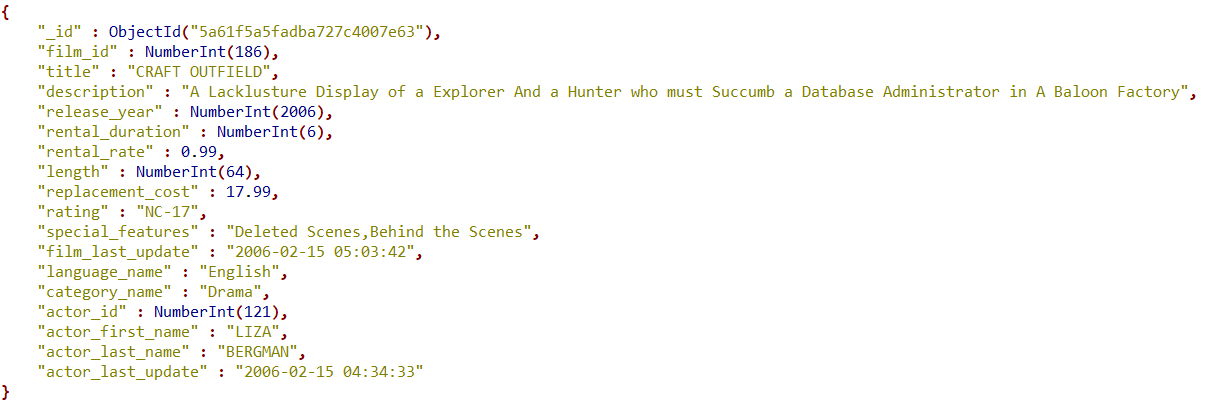
**]**

**}**

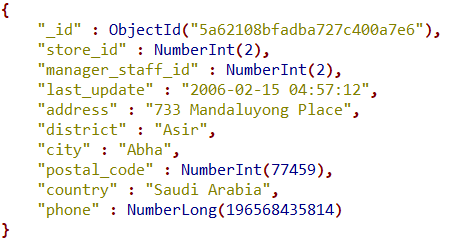
Exemple d’objet de la table **customer**



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 **staff**

