Léna BAILLET- Louis GREINER - Benjamin MISSAOUI - Rodolphe OLSOMMER

NF18 **Projet**

Gestion d'une base de données de spectacles culturels

NORMALISATION D'UNE BASE DE DONNÉES RELATIONNELLE ET TRANSFORMATION EN NOSQL (MONGODB)

ALLIANCE SORBONNE UNIVERSITÉ | utc

P21

TABLE DES MATIÈRES

1. NORMALISATION

Les formes normales ont pour objectif de définir la décomposition des schémas relationnels, tout en préservant les DF et sans perdre d'informations, afin de représenter les objets et associations canoniques du monde réel de façon non redondante. L'objectif de la décomposition est de "casser" une relation en relations plus petites afin d'en éliminer les redondances et sans perdre d'information.

Notre base de données est déjà en 3NF:

- 1NF : Toutes les relations possèdent au moins une clé et tous leurs attributs sont atomiques,
- 2NF : Pour les classes possédant des clés candidates composées, comme la relation salle, tout attribut dépend de la clé entière et non seulement d'une seule de ses parties.
- 3NF: Tout attribut n'appartenant à aucune clé candidate ne dépend directement que de clés candidates

1.1 ~ Failles restantes de notre base en 3NF ~

Afin de rendre notre base de données plus performante, il est important de guider l'utilisateur lors de la saisie de données. Un objet peut-être décrit de différente façon, mais dans la BDD une saisie qui diffère créera simplement un ojet différent. Par exemple: la personne "Jules César" sera différente de "Jules Cesar". Dans l'optique d'éviter ces anomalies de données, nous avons décomposé les tables Association, Concert, Categorie, Exterieur.

Jusqu'à présent, nous ne contrôlions pas la catégorie de l'association, et l'utilisateur pouvait donc saisir "évènementiel" ou "evenement", ce qui aurait posé par exemple des difficultés pour retrouver toutes les associations du pôle évènementiel.

1.2 ~ CORRECTION DU MLD ~

Voici le résultat des décompositions:

Association(#nom:string, description: string, catégorie: string, email: string, dateCréation: date, siteWeb: string) avec email UNIQUE, siteWeb NULLABLE
devient:
Association(#nom:string, description: string, catégorie=> categorieAsso, email: string, dateCréation: date, siteWeb: string) avec email UNIQUE, siteWeb NULLABLE
categorieAsso(#categorie : string)

Extérieur(#idPersonne=>Personne, organisme: string, contact: string) avec contact KEY
devient:

Extérieur(#idPersonne=>Personne, organisme => Organisme, contact: string) avec contact KEY
Organisme(#organisme: string)

Concert(#idConcert=>Spectacle, compositeur: string, annéeParution: date, genre: string)
devient:
Concert(#idConcert=>Spectacle, compositeur: string, annéeParution: date, genre =>genreConcert)
genreConcert(#genre: string)

Categorie(#idSeance => Seance, #idBillet => Billet, categorie: string)
devient:
categorie(#idSeance => Seance, #idBillet => Billet, categorie => categorieBillet)

categorieBillet(#categorie: string)

2. CONSTRUCTION D'UNE BASE DE DONNES NON RELATIONNELLE AVEC MONGODB, COUCHE APPLICATIVE PYTHON

Nous avons choisi de transformer notre base de données relationnelle en base de données non-relationnelle, et de l'implémenter avec MongoDB, qui stocke les données au format JSON. L'architecture d'une base de donnée MongoDB est la suivante:

- Database
    - Collections (anciennement table dans les base de données relationnelles)
        - Document au format JSON (anciennement enregistrement d'une table dans une base de données relationnelle)

Les bases MongoDB sont des bases dites schema-less. Une collection peut contenir des documents de structures différentes et il n'est pas possible de définir la structure a priori d'une collection. La structure d'une collection n'est donc définie que par les documents qui la composent, et elle peut évoluer dynamiquement au fur et à mesure des insertions et suppressions.

2.1 ~ Choix des tables ~

Pour répondre au mieux aux besoins du sujet, nous avons décidé de garder, pour conserver les mêmes fonctionnalités dans la couche applicative, les tables suivantes:

- Association
    - nom
    - description
    - categorie
    - email
    - dateCreation

    - siteWeb
    - president (objet JSON de la forme des documents dans la collection Personne)
    - tresorier (objet JSON de la forme des documents dans la collection Personne)
    - membres (un tableau d'objets JSON de la forme des documents dans la collection Personne)

- Billet
    - dateCreation
    - tarif
    - categorie
    - seance (objet JSON de la forme des documents dans la collection Seance)

- Personne
    - id
    - prenom
    - nom
    - type
    - informations
        - numCin (si type = étudiant ou personnel)
        - statut (si type = personnel)
        - organisme (si type = extérieur)
        - contact (si type = extérieur)

- Réservation
    - date
    - heure
    - association (objet JSON de la forme des documents dans la collection Association)
    - salle (objet JSON de la forme des documents dans la collection Salle)

- Salle
    - numSalle
    - capacite
    - numBat
    - typeSalle

- Séance
    - id
    - date
    - spectacle (objet JSON de la forme des documents dans la collection Spectacle)
    - salle (objet JSON de la forme des documents dans la collection Salle)

- Spectacle
    - id
    - duree
    - association (objet JSON de la forme des documents dans la collection Association)
    - informations
        - genre
        - auteur (si type = Théâtre)
        - anneeParution (si type = Théâtre ou Concert)
        - Compositeur (si type = Concert)

Nous n'avons, en non-relationnel, évidemment pas besoin des anciennes tables classes-associations, et des tables créées par les différentes classes héritages.

2.2 ~ Implémentation de la base via MongoDB Compass ~

Pour l'implémentation de la base via MongoDB Compass, nous avons créé des fichier JSON (en suivant la syntaxe ci-après détaillée) pour chacune des collections. Nous avons ensuite créé une base de données sur MongoDB Compass, créé les collections puis importé dans chacune d'elle leur fichier JSON correspondant.

```
[
    {
        "nom": "Etuville",
        "description": "Organisation du gala de l'UTC et de l'Utcéenne",
        "categorie": "Evènementiel",
        "email": "etuville@asso.utc.fr",
        "dateCreation": "2005-01-01",
        "siteWeb": "https://assos.utc.fr/etuville/",
        "president":
            {
                "id": 1,
                "prenom": "Louis",
                "nom": "Greiner",
                "type": "etudiant",
                "informations": {
                    "numCIN": 234
                }
            },
        "tresorier":
            {
                "id": 4,
                "prenom": "Benjamin",
                "nom": "Missaoui",
                "type": "etudiant",
                "informations": {
                    "numCIN": 750
                }
            },
        "membres":
            [
                {
                    "id": 3,
                    "prenom": "Léna",
                    "nom": "Baillet",
                    "type": "etudiant",
                    "informations": {
                        "numCIN": 122
                    }
                }
            ]
    }
]
```


2.3 ~ Couche applicative avec la libraire pymongo ~

Bien que cela ne soit pas demandé ici, voici tout de même un exemple d'insertion d'un nouvel enregistrement avec Python.

```python
# Import de pymongo pour faire les requêtes
from pymongo import MongoClient
client = MongoClient('mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb')

# Choix de la BDD
db = client.NF18_nosql

# Déf du nouvel utilisateur
newUser = {
    "id": 7,
    "prenom": "Fatma",
    "nom": "Chamekh",
    "type" : "personnel",
    "informations" : {
        "numCIN" : 7913
    }
}

# Insert du nouvel utilisateur
result = db.personne.insert_one(newUser)
```