UNIVERSITÉ DE TECHNOLOGIE DE COMPIÈGNE - NF18

Système de gestion pour une bibliothèque

Rapport

Projet NF18 TD1_G2 Bibliothèque

Groupe : TD1_G2

- **Pillis Julien**
- **Galleze Rayane**
- **Alexandre Laboure**
- **Benjamin Vivat**

Sommaire

Introduction

On souhaite concevoir un système de gestion pour une bibliothèque municipale qui souhaite informatiser ses activités : catalogage, consultations, gestion des utilisateurs, prêts, etc., dont les besoins sont :

- Faciliter aux adhérents la recherche de documents et la gestion de leurs emprunts.
- Faciliter la gestion des ressources documentaires : ajouter des documents, modifier leur description, ajouter des exemplaires d'un document, etc.
- Faciliter au personnel la gestion des prêts, des retards et des réservations.
- Faciliter la gestion des utilisateurs et de leurs données.
- Établir des statistiques sur les documents empruntés par les adhérents, cela permettra par exemple d'établir la liste des documents populaires, mais aussi d'étudier le profil des adhérents pour pouvoir leur suggérer des documents.

Dans ce rapport, nous détaillerons toutes les étapes de la solution proposée ; de la conception de la base de données à la réalisation de l'application python permettant de la peupler et l'utiliser.

Note de Clarification

Pour avoir une meilleure vision du problème, nous établissons dans un premier temps une note de clarification :

RESSOURCE

- code (clé), titre, date d'apparition, code de classification, éditeur, un genre
- a un type (RESSOURCE est une classe mère (abstraite), héritage exclusif non complet, avec LIVRE, FILM, MUSIQUE par référence avec contrainte de projection et intersection des classes filles nulle?)

CONTRIBUTEUR

- nom, prénom, date de naissance, nationalité
- un contributeur peut composer 1 ou plusieurs musiques (compositeur) (association avec MUSIQUE "1..n" -- "*")
- un contributeur peut interpréter 1 ou plusieurs musiques (interprète) (association avec MUSIQUE "1..n" -- "*")
- un contributeur peut réaliser 1 ou plusieurs films (réalisateur) (association avec FILM "1..n" -- "*")
- un contributeur peut jouer dans 1 ou plusieurs films (acteur) (association avec FILM "1..n" -- "*")
- un contributeur peut écrire 1 ou plusieurs livres (auteur) (association avec LIVRE "1..n" -- "*")
- clé artificielle à envisager pour éviter les homonymes

NB: Nous pourrons aussi envisager un héritage avec la classe mère CONTRIBUTEUR et des classes filles ACTEUR, COMPOSITEUR, REALISATEUR, INTERPRETE, AUTEUR à voir...

LIVRE (classe fille de RESSOURCE)

+ attributs de la classe mère RESSOURCE

- ISBN, résumé, langue
- a été écrit par un ou plusieurs contributeurs (association avec CONTRIBUTEUR * -- 1..n)

FILM (classe fille de RESSOURCE)

+ attributs de la classe mère RESSOURCE

- synopsis, langue, longueur
- a été réalisé par un ou plusieurs réalisateurs (association avec CONTRIBUTEUR : * -- 1..n)
- a été joué par un ou plusieurs acteurs (association avec CONTRIBUTEUR : * -- 1..n)

MUSIQUE (classe fille de RESSOURCE)

+ attributs de la classe mère RESSOURCE

- longueur
- a été composée par un ou plusieurs compositeurs (association avec CONTRIBUTEUR : * -- 1..n)
- est interprétée par un ou plusieurs musiciens (association avec CONTRIBUTEUR : * -- 1..n)

UTILISATEUR

- nom, prénom, adresse e-mail, adresse (composée du numéro, de la rue, du code postal et de la ville)
- a un compte utilisateur (Composition 1 -- 1 avec COMPTE)
  Pour des raisons de sécurité, il serait préférable d'avoir une table Compte à part avec le login et mot de passe.
- clé artificielle à envisager pour éviter les homonymes
- chaque utilisateur dispose de droits spécifiques sur la BDD en fonction de leur rôle exemples:
  - L'admin a le rôle de superuser (dispose de tous les droits) *grant all privileges*
  - Une bibliothécaire peut modifier le statut des adhérents, ajouter des sanctions, ajouter des ressources ect...
  - Un adhérent peut consulter les ressources disponibles et gérer ses emprunts ect...

PERSONNEL (classe fille d'UTILISATEUR)

\+ attributs de la classe mère UTILISATEUR

ADHÉRENT (classe fille d'UTILISATEUR)

\+ attributs de la classe mère UTILISATEUR

- date de naissance, numéro de tél
- statut_adhésion ENUM(active, expirée, suspendue, blacklistée)
- Pour supprimer un adhérent, il suffit de rendre son statut expiré vu que : "*La bibliothèque souhaite garder trace de toutes les adhésions, actuelles et passées.*"
- peut emprunter plusieurs exemplaires (association avec Exemplaire : * -- *)

SANCTION

- Motif ENUM(retard, détérioration, perte)
- Date de début (not NULL)
- Date de fin (peut être NULL en cas de suspension indéterminée)
  "*En cas de perte ou détérioration grave d'un document, la suspension du droit de prêt est maintenue jusqu'à ce que l'adhérent rembourse le document*"
- est liée à un adhérent (association avec ADHERENT : "*" -- "1")
- date de début <= date de fin

COMPTE

- login, mot de passe
- est lié à un UTILISATEUR (Composition)

PRÊT (association entre Adhérent et Exemplaire)

- date de début, date de fin, rendu (booléen)
    - Choix d'une date de fin plutôt qu'une durée, car plus simple à gérer au niveau applicatif et la durée est retrouvable à partir de la date de début et de fin.
    - Le rendu permet de savoir si la ressource a été ramenée et vérifier s'il y a un retard ou non
- date de début <= date de fin

EXEMPLAIRE

- état ENUM(Neuf, Bon, Abîmé, Perdu))
- codeRessource (composition avec Ressource 1..*, un exemplaire ne peut pas exister sans ressource),
- disponible (Booléen)
- clé artificielle à envisager

REQUÊTES TYPES

- Établir une liste des documents classés par popularité
- Suggérer une liste de recommandations à un adhérent en fonction de ses précédents prêts
- Classer les ressources par contributeur, genre, date…
- Déterminer le nombre de sanction d'un adhérent
- Calculer le nombre d'adhésions sur une période donnée (faire une moyenne de variation par année, par exemple en 2022 on a 30% d'adhésions en plus par rapport à 2021)

Modèle Conceptuel de Données

De cette note de clarification, nous pouvons désormais construire le diagramme UML correspondant (MCD) :

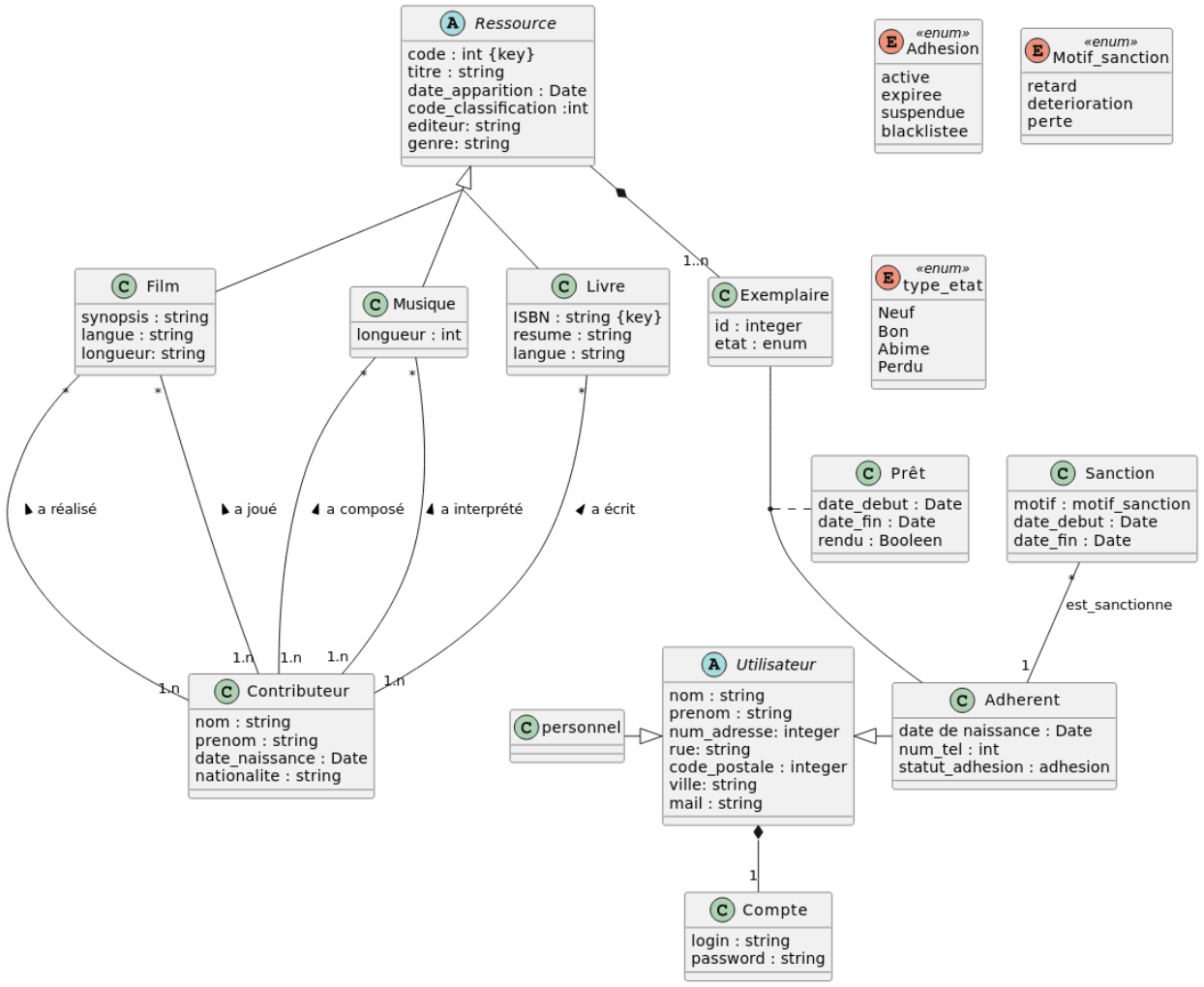

Figure 1 : Modèle conceptuel de données avec UML

Modèle Logique de Données

Après avoir établi un MCD, nous en avons déduit un MLD dont les tables sont détaillées ci-dessous. Nous avons également ajouté les contraintes associées aux héritages.

A. Tables

Ressource(#code: int, titre: string, date_apparition: date, code_classification: int, editeur: string, genre: string) avec titre, code_classification NOT NULL

- Classe mère abstraite, héritage exclusif : une ressource a un type unique (Film ou Musique ou Livre), héritage non complet (les classes filles ont des attributs propres à elles).
- Héritage par classe mère à éviter car il y aurait beaucoup de contraintes à cause des associations des classes filles. Dde plus, ce type d'héritage implique l'ajout de contraintes de vérification des types (par exemple, lorsqu'on instancie un film, on doit s'assurer que les attributs ISBN et résumé sont NULL).
- Héritage par classe filles à éviter car la classe mère possède une composition avec exemplaire et l'héritage est exclusif.
- => On opte ainsi pour un héritage par référence.

Film(#code=>Ressource.code: int, synopsis: string, langue: string, longueur: int) avec langue, synopsis, longueur NOT NULL

Musique(#code=>Ressource.code: int, longueur: int) avec longeur NOT NULL

Livre(#code=>Ressource.code, ISBN: int, resume: texte, langue: string) avec ISBN key et resume, langue NOT NULL

Acteur(#film=>Film.code: int, #contrib=>Contributeur.id: int)

Realisateur(#film=>Film.code: int, #contrib=>Contributeur.id: int)

Auteur(#livre=>Livre.code: int, #contrib=>Contributeur.id: int)

Compositeur(#musique=>Musique.code: int, #contrib=>Contributeur.id: int)

Interprete(#musique=>Musique.code: int, #contrib=>Contributeur.id: int)

Contributeur(#id: int, nom: string, prenom: string, date_naissance: date, nationalite: string) avec * NOT NULL

Exemplaire(#id: int, #code_ressource=>Ressource.code: int, etat:{neuf, bon, abime, perdu}) avec etat NOT NULL

Pret(#id_pret:int, adherent=>Adherent.id: int, code_ressource=>Exemplaire.code_ressource, exemplaire=>Exemplaire.id: int, date_debut: date, date_fin: date, rendu: booleen) avec date_debut <= date_fin, all NOT NULL

Sanction(#idSanction: int, idAdherent=>Adherent.id: int, motif: {retard, deterioration, perte}, date_debut: date, date_fin: date) avec date_debut <= date_fin, date_debut, motif NOT NULL

> "*Si une clé candidate (globale) permet d'identifier de façon unique une partie indépendamment du tout, on préférera la conserver comme clé candidate plutôt que de la prendre pour clé primaire. Si on la choisit comme clé primaire cela revient à avoir transformé la composition en agrégation, en redonnant une vie propre aux objets composants.*" D'après le cours mod4 sur la transformation des compositions en relationnelle, il est donc, ici, plus judicieux de choisir login en tant que key (unique not null) et non en tant que clé primaire

Utilisateur(#id: int, nom: string, prenom: string, num_adresse: int, rue: string, code_postal: int, ville: string, mail: string) nom, prenom not null

- Classe mère abstraite, héritage non complet (les classes filles ont des attributs propres à elles).
- Héritage par classe mère à éviter car il y aurait beaucoup de contraintes à causes des associations des classes filles. De plus ce type d'héritage implique l'ajout de contraintes de vérification des types (par exemple, lorsqu'on instancie un utilisateur (personnel), on doit s'assurer que le statut d'adhésion est NULL).
- Héritage par classe filles à éviter car la classe mère possède une composition avec compte.
- => On opte ainsi pour un héritage par référence.

Personnel(#id=>Utilisateur.id: int)

Adherent(#id=>Utilisateur.id: int, date_naissance: date, num_tel: int, statut_adhesion: {active, expiree, suspendue, blacklistee}) statut_adhesion NOT NULL

Compte(#id=>Utilisateur.id: int, login: string, password: string) login key, password not null

B. Contraintes

Il est indispensable de ne pas oublier les contraintes associées aux héritages :

Héritage classe Ressource :

- INTERSECTION (PROJECTION(Film,code), PROJECTION(Musique,code), PROJECTION(Livre,code)) = NULL (Héritage exclusif classe Ressource)

- UNION (PROJECTION(Film,code), PROJECTION(Musique,code), PROJECTION(Livre,code)) = PROJECTION(Ressource,code) (classe Ressource abstraite)

Héritage classe Utilisateur :

- INTERSECTION (PROJECTION(Personnel,id), PROJECTION(Adherent,code)) = NULL (Héritage exclusif classe Utilisateur)

- UNION (PROJECTION(Personnel,id), PROJECTION(Adherent,id), = PROJECTION(Utilisateur,id) (classe Utilisateur abstraite)

Application python

Nous avons réalisé une application python avec une interface graphique, permettant à un utilisateur d'interagir pleinement avec notre base de données. Ainsi, l'utilisateur pourra se connecter, en tant qu'adhérent ou en tant que membre du personnel de la bibliothèque, et effectuer un certain nombre d'actions en fonction de son rôle.

Figure 2 : Capture d'écran du menu de l'application

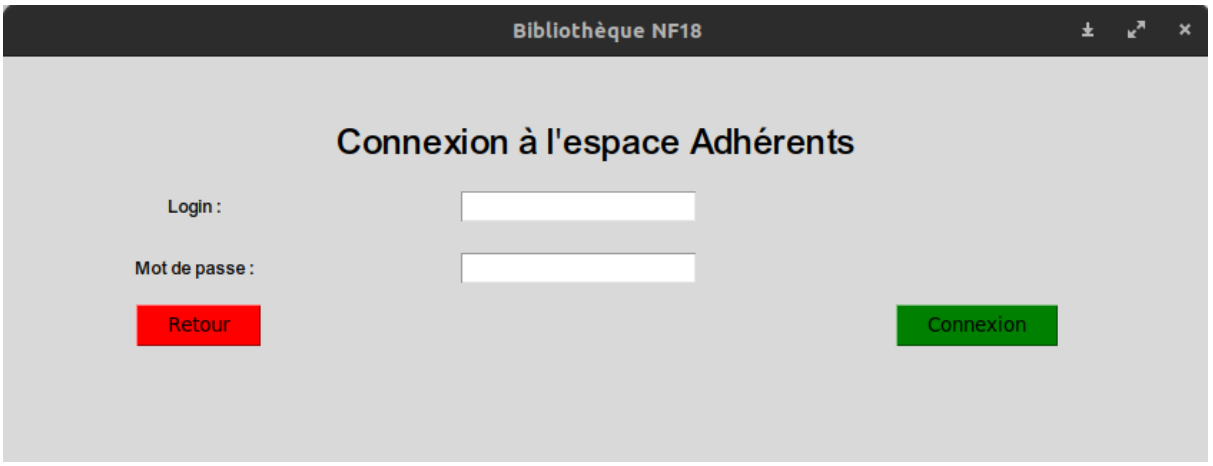

Figure 3 : Capture d'écran de l'interface de connexion des adhérents à leur espace

Figure 4 : Capture d'écran de l'interface de connexion du personnel

En tant qu'adhérent, on pourra visualiser ses informations personnelles, ses sanctions, mais aussi afficher toutes les ressources qui sont disponibles, triées par type de ressource (Films, Musiques, Livres) ou par contributeur. On pourra ensuite afficher des détails concernant une ressource qui nous intéresse.

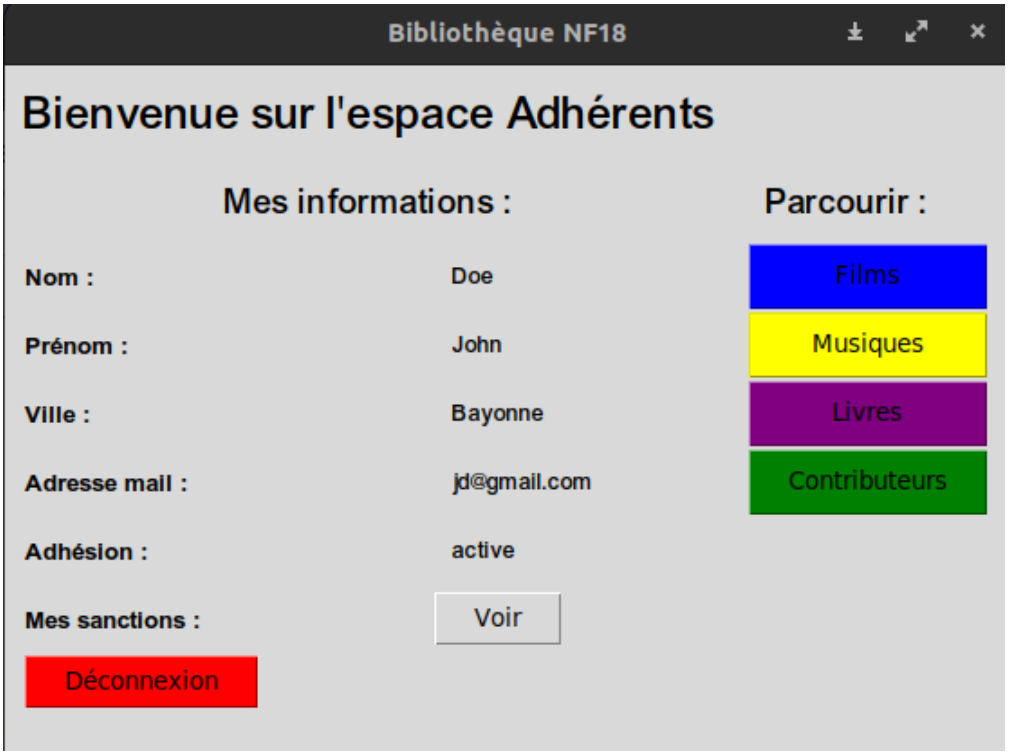

Figure 5 : Capture d'écran de l'interface de l'espace d'un adhérent

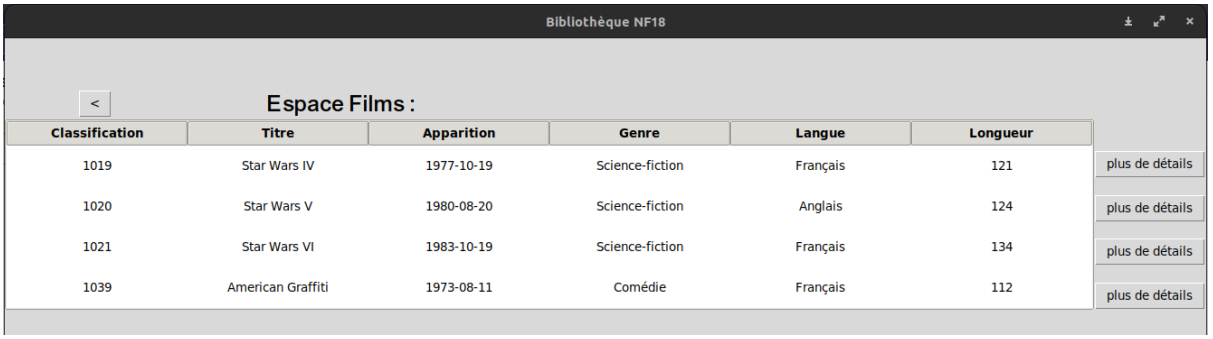

Figure 6 : Capture d'écran de l'interface de visualisation des films

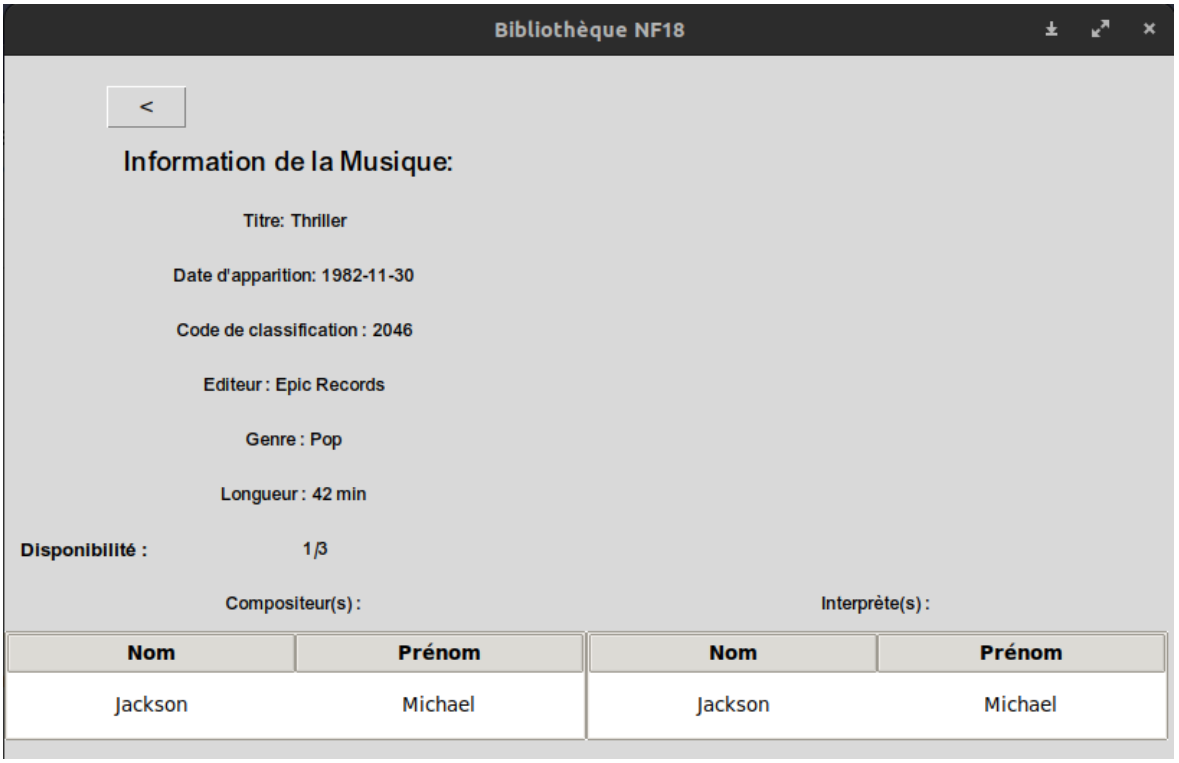

Figure 7 : Capture d'écran de l'interface de visualisation des informations sur une musique

En tant que membre du personnel, on pourra ajouter un adhérent, une ressource, et des exemplaires, mais aussi enregistrer un prêt pour un exemplaire et un adhérent donné, ainsi qu'une sanction si nécessaire. On accèdera aussi à quelques informations complémentaires et statistiques sur la page d'accueil de l'application, telles que le nombre de prêts en cours, la durée moyenne d'un prêt ou les prêts actuellement en retard.

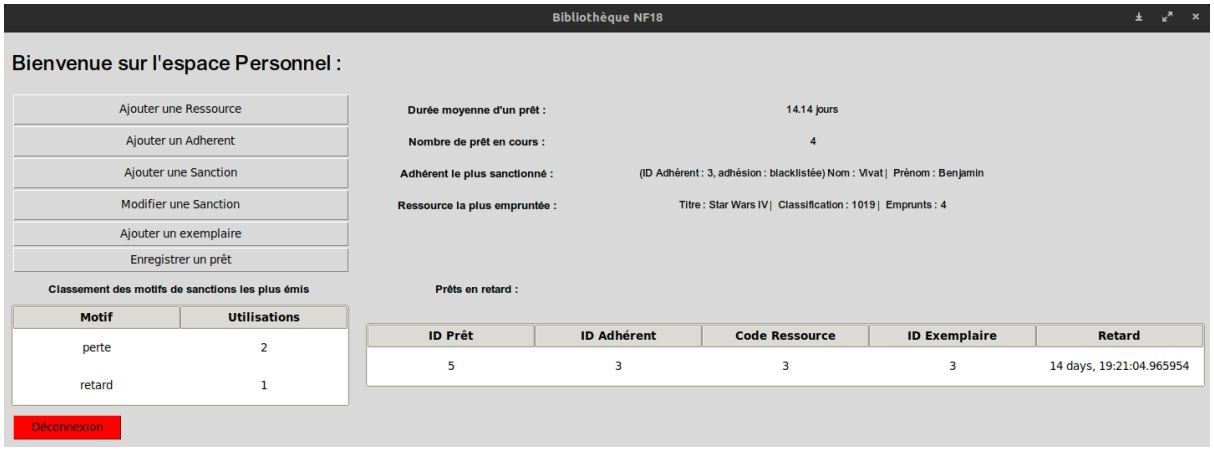

Figure 7 : Capture d'écran de l'interface de l'espace d'un membre du personnel

Le code complet de l'application python est joint à ce rapport.

Exemples de requêtes

Voici quelques exemples de requêtes statistiques implémentées dans l'application Python :

```
/*Durée moyenne d'un emprunt*/
query = "SELECT AVG(date_fin - date_debut) AS duree_moyenne FROM Pret;"

/*Nombre d'emprunts en cours */
 query = "SELECT COUNT(*) FROM Pret
          WHERE NOT rendu;"

/*Ressource la plus empruntée*/
query = "SELECT idadherent, prenom, nom, statut_adhesion, COUNT(idadherent)
AS nombre_sanctions FROM Sanction S
         INNER JOIN Adherent A ON S.idadherent = A.id
         NATURAL JOIN Utilisateur
         GROUP BY idadherent, statut_adhesion, prenom, nom
         ORDER BY nombre_sanctions;"

/*Adhérent le plus sanctionné*/
query = "SELECT titre, code_classification, COUNT(code_ressource) AS
nb_emprunts FROM Ressource R
         INNER JOIN Pret P on R.code = P.code_ressource
         GROUP BY code_ressource, titre, code_classification
         ORDER BY nb_emprunts
         DESC LIMIT 1;"

/*Classement des motifs de sanctions les plus récurrents*/
query = "SELECT motif, COUNT(motif) AS nombre FROM Sanction
         GROUP BY motif
         ORDER BY nombre
         DESC;"

/*Liste des prêts dont le rendu est en retard*/
query = "SELECT id_pret, adherent, code_ressource, exemplaire, NOW() -
date_fin AS retard FROM Pret
         WHERE date_fin < NOW() AND NOT rendu;"
```

Implémentation JSON

I. Réflexion sur l'intégration d'attributs de type JSON

Dans cette étape de notre conception d'une base de données, nous allons nous intéresser à l'implémentation d'attributs de type JSON dans nos tables déjà conçues. Pour ce faire, regardons de plus près notre diagramme UML :

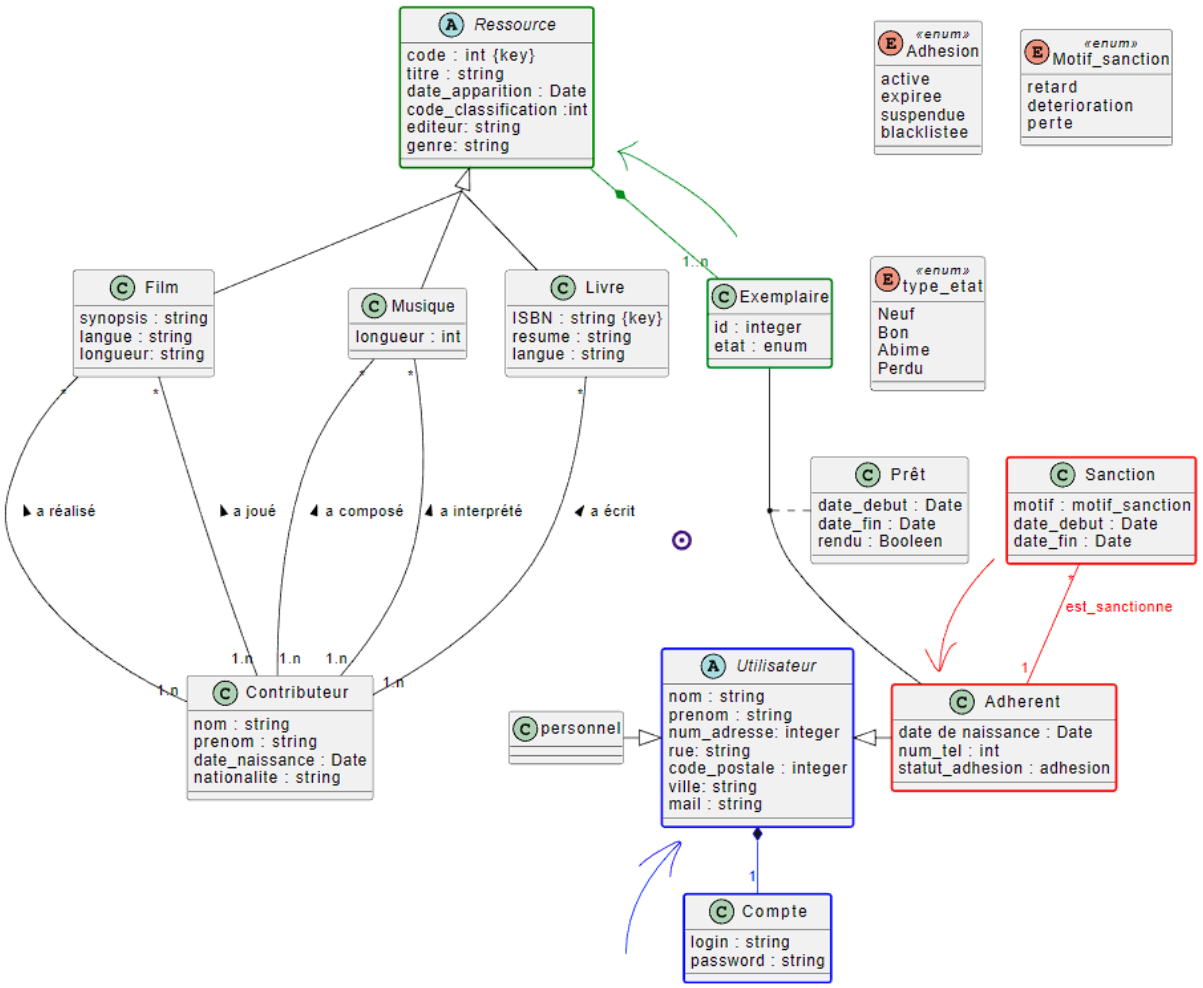

Figure 8 : Evolution R-JSON du diagramme de classe

Nous observons 3 associations (zones rouge, bleue et verte) dans que nous pouvons modifier à l'aide du type JSON.

En zone verte , la relation **EXAMPLAIRE** peut se transformer en attribut JSON au sein de la relation **RESSOURCE** car il s'agit d'une composition.

Cependant, nous devons faire attention à bien conserver l'attribut **id** d'exemplaire, car nécessaire pour son association avec la classe **ADHERENT** (classe d'association **PRET**).

Par contrainte de la cardinalité **1…n**, nous devons ajouter la contrainte **NOT NULL** à ce nouvel attribut.

Cependant, cette implémentation est pour le moment difficilement envisageable. En effet, n'ayant pas vu cela dans le cours, nous ne savons pas comment récupérer l'id de l'exemple, qui est un sous-attribut d'un attribut json

Dans la suite de cette partie, nous avons supposé qu'il est possible de récupérer cet id.

En zone rouge, la relation **SANCTION** peut se transformer en attribut JSON au sein de la relation **ADHERENT** grâce aux cardinalités de l'association (équivalent à une composition).

Enfin, en zone bleue, la relation **COMPTE** peut se transformer en attribut JSON au sein de la relation **UTILISATEUR** car il s'agit d'une nouvelle fois d'une composition.

Par contrainte de la cardinalité **1**, nous devons ajouter la contrainte **NOT NULL** à ce nouvel attribut.

II. Implémentation de ces attributs dans les tables concernées (SQL)

```sql
CREATE TABLE Ressource(
    code INTEGER PRIMARY KEY,
    titre VARCHAR NOT NULL,
    date_apparition DATE,
    code_classification INTEGER NOT NULL,
    editeur VARCHAR,
    genre VARCHAR,
    exemplaires JSON NOT NULL
);


CREATE TABLE Utilisateur(
    id SERIAL PRIMARY KEY ,
    nom VARCHAR NOT NULL,
    prenom VARCHAR NOT NULL,
    num_adresse INTEGER,
    rue VARCHAR,
    code_postal INTEGER,
    ville VARCHAR,
    mail VARCHAR,
    compte JSON NOT NULL
);
```

```
CREATE TABLE Adherent(
    id INTEGER REFERENCES Utilisateur(id),
    date_naissance DATE,
    num_tel INTEGER,
    statut_adhesion Adhesion NOT NULL,
    sanctions JSON,
    PRIMARY KEY(id)
);


CREATE TABLE Pret(
    id_pret SERIAL PRIMARY KEY ,
    adherent INTEGER REFERENCES Adherent(id) NOT NULL,
    exemplaire INTEGER REFERENCES Ressource(exemplaire.id),
    code_ressource INTEGER REFERENCES Ressource(code),
    date_debut DATE NOT NULL,
    date_fin DATE ,
    rendu BOOLEAN NOT NULL,

    CHECK(date_fin>=date_debut)
);
```

Remarque : Dans la création de la table **PRET** nous supposons qu'exemplaire récupère l'id de l'exemplaire.

(c.f : Partie I)

III. Exemples d'insertion sur ces nouvelles tables

```
INSERT INTO utilisateur(nom, prenom, num_adresse, rue, code_postal,
ville, mail, compte) values(
    'Galleze',
    'Rayane',
    '21',
    'rue du dépôt',
    '60280',
    'Compiègne',
    'rgalleze@etu.utc.fr',
    '{"login":"rgalleze", "password":"2908$RGnf18tropbien"}'
);

INSERT INTO adherent values(
    3,
```

```
    '2000-02-02',
    0668234125,
    'blacklisté',
    '[

{"motif":"perte","date_debut":"2022-11-03","date_fin":"2022-11-05"},
        {"motif":"perte","date_debut":"2022-11-11","date_fin":null}
    ]'
);


INSERT INTO Ressource VALUES(
    3,
    'The Lord of the Rings',
    '1954-07-29',
    1154,
    'Allen & Unwin',
    'Fantasy',
    '[
        {"id":"1","etat":"abîmé"},
        {"id":"2","etat":"bon"},
        {"id":"3","etat":"abîmé"},
        {"id":"4","etat":"abîmé"}
    ]'
);
```

IV. Exemples requêtes avec implémentation JSON

```
SELECT nom, prenom, num_adresse, rue, code_postal, ville, mail,
c->>'login' AS login, c->>'password' AS password
FROM Utilisateur u, JSON_ARRAY_ELEMENTS(u.compte) c;

SELECT u.nom, u.prenom, a.statut_adhesion, s->>'motif' AS
motif_sanction, CAST(s->>'date_debut' AS DATE) AS debut_sanction,
CAST(s->>'date_fin' AS DATE) AS fin_sanction
FROM Adherent a NATURAL JOIN Utilisateur u,
JSON_ARRAY_ELEMENTS(a.sanctions) s;

SELECT code, titre, editeur, CAST(e->>'id' AS INTEGER) AS id_exemplaire,
e->>'etat' AS etat
FROM Ressource r, JSON_ARRAY_ELEMENTS(r.exmplaires) e;
```

Conclusion

Pour conclure ce rapport, nous pouvons dire que nous avons normalement réussi à mettre en place une base de données permettant de répondre au problème posé. Les requêtes construites permettent de répondre aux besoins de la bibliothèque et le système conçu permet de gérer efficacement tant les ressources que les utilisateurs et leurs prêts.
De la conception à la réalisation et à l'optimisation JSON, ce projet nous aura permis de mieux comprendre certains concepts, d'en mettre en application un bon nombre de ces concepts, et d'améliorer nos connaissances sur la conception de base de données.