20 mars 2025

GROUPE 12

PROJET - MA BIBLIOTHÈQUE

IFT2935 - Base de données

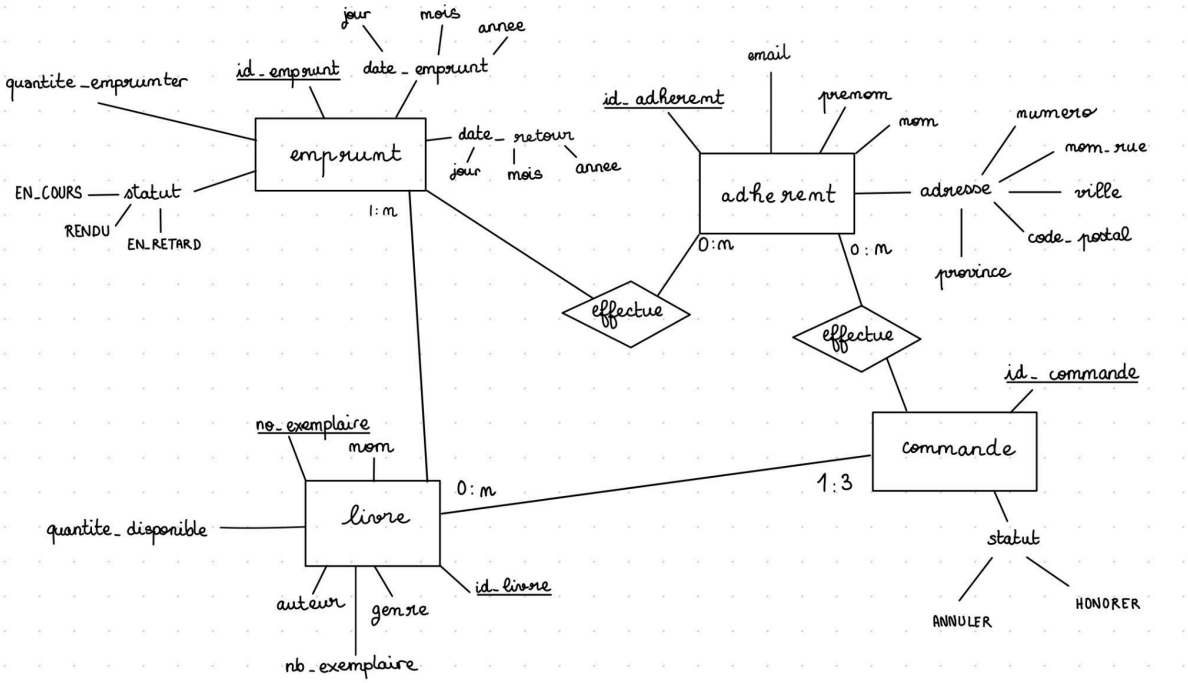
HAYS Oceane

CHEN Xi

BELLOTTI Maxime

AKAY Derin

1. MODÉLISATION ENTITÉ/ASSOCIATION



** la date_empunt est la date_debut

- CONTRAINTES D'INTÉGRITÉS -

- Un livre ne peut pas être rendu le même jour que son emprunt.

$\forall x \in Emprunt \rightarrow x.date\_debut \neq date\_retour$

- La durée maximum d'un emprunt est de 14 jours.

$\forall x \in Emprunt \rightarrow x.date\_debut - date\_retour \leq 14$

- Un adhérent peut commander au maximum 3 livres par commande.

- EXPLICATION -

Adhérent

Pour cette entité, nous avons le ID unique identifiant chaque adhérent, le email unique , afin de contacter l'adhérent par exemple en cas de retard ou si sa commande est disponible ou bien pour la newsletter. Une adresse complète pour vérifier qu'il est bien associé à la bibliothèque de son secteur

Emprunt

Un adhérent peut effectuer plusieurs emprunts mais il doit ramener les livres empruntés sous 14 jours, on garde donc la date d'emprunt et on attend pour une date de retour en défaut null, passer ce délai, le statut passe à EN RETARD. On conserve le nombre de livres empruntés.

Commande

On garde le ID de la commande et on suppose qu'il ne peut commander que trois livres au maximum à chaque commande. Un livre peut apparaître dans plusieurs commandes.

Livre
Plusieurs exemplaires du même livre peuvent exister dans une bibliothèque donc on garde l' ID du livre et le numéro de l'exemplaire, ainsi que la quantité disponible.

2. LA TRANSFORMATION

Livre (id_livre, no_exemplaire, titre, #nom_genre, auteurs, quantite_disponible, langue, edition)
Adhérent(id_adherent, prenom, nom, email, no_rue, nom_rue, ville, province, pays)
Emprunt (id_emprunt, #id_adherent, date_debut, date_fin, #statut_emprunt, #no_exemplaire, #id_livre)
Commande (id_commande, #id_adherent, #statut_commande, #id_livre)
Genre(genre)

3. LA NORMALISATION

Dépendances Fonctionnelles + Normalisation

Livre { id_livre → titre, nom_genre, auteur, langue, edition, no_exemplaire }

Est-ce 1NF ? non car un livre peut avoir plusieurs exemplaires, et plusieurs auteurs, nous créons donc la relation
Exemplaire (no_exemplaire, #id_livre, disponibilité)
Exemplaire { no_exemplaire, id_livre → disponibilité }
Auteur(id_auteur, prénom, nom, nationalité, annee_naissance)
Auteur { id_auteur → prénom, nom, nationalité, annee_naissance }
Livre_Auteur(#id_livre, #id_auteur)
Livre (id_livre, titre, genre, langue, edition)
Livre { id_livre → titre, genre, langue, edition }

Est-ce 2NF ? oui, pas de dépendance partielle
Est-ce 3NF ? oui car les attributs dépendent directement de la clé.
Peut-on utiliser Boyce-Codd ? Tous les attributs clés dont a droite et non-clés sont à gauche dont c'est BCNF.

Adhérent { id_adherent → prenom, nom, email, no_rue, nom_rue, ville, province, pays }

Est-ce 1NF ? oui, pas d'attributs multi-valués.
Est-ce 2NF ? oui
Est-ce 3NF ? oui
Peut-on utiliser Boyce-Codd ? oui
Pas de décomposition nécessaire.

Emprunt {id_emprunt → id_adherent, date_debut, date_fin, statut_emprunt, no_exemplaire, id_livre}

Est-ce 1NF ? oui
Est-ce 2NF ? oui

Est-ce 3NF ? oui
Peut-on utiliser Boyce-Codd ? oui

Pas de décomposition nécessaire.

Commande { id_commande → id_adherent, statut_commande, id_livre }

Est-ce 1NF ? oui
Est-ce 2NF ? oui
Est-ce 3NF ? oui
Peut-on utiliser Boyce-Codd ?

Pas de décomposition nécessaire.

RÉSULTAT - Relations + Dépendances fonctionnelles associées

Livre (id_livre, titre, genre, langue, edition)
Exemplaire (no_exemplaire, #id_livre, disponibilité)
Auteur(id_auteur, prénom, nom, nationalité, annee_naissance)
Livre_Auteur(#id_livre, #id_auteur)
Adhérent(id_adherent, prenom, nom, email, no_rue, nom_rue, ville, province, pays)
Emprunt (id_emprunt, #id_adherent, date_debut, date_fin, #statut_emprunt, #no_exemplaire, #id_livre)
Commande (id_commande, #id_adherent, #statut_commande, #id_livre)
Statut_Commande(statut_commande)
Statut_Emprunt(statut_emprunt)
Genre(Genre)

Livre { id_livre → titre, genre, langue, edition }
Exemplaire { id_livre, no_exemplaire → disponibilité }
Auteur { id_auteur → prénom, nom, nationalité, annee_naissance }
Adhérent { id_adherent → prenom, nom, email, no_rue, nom_rue, ville, province, pays }
Commande { id_commande → id_adherent, statut_commande, id_livre }
Emprunt {id_emprunt → id_adherent, date_debut, date_fin, statut_emprunt, no_exemplaire, id_livre}

4. L'IMPLÉMENTATION

5. NOS QUESTIONS :

Question 1 : *Quels sont les livres actuellement empruntés et non retournés ?*

SQL :

```
SELECT l.titre, a.prenom, a.nom, e.date_debut, e.date_fin
FROM Emprunt e
JOIN Livre l ON e.id_livre = l.id_livre
```

```
JOIN Adherent a ON e.id_adherent = a.id_adherent
WHERE e.statut_emprunt != 'rendu';
```

Requête optimisée en algèbre relationnelle :

r1 ← σ_*{statut_emprunt ≠ 'rendu'}* (Emprunt)
r2 ← π_*{id_adherent, id_livre, no_exemplaire, date_debut, date_fin}* (r1)
r3 ← π_*{id_livre, titre}* (Livre)
r4 ← r2 ⋈ r3
r5 ← π_*{id_adherent, titre, date_debut, date_fin}* (r5)
r6 ← π_*{id_adherent, prenom, nom}* (Adherent)
r7 ← r5 ⋈ r6
Resultat ← π_*{titre, prenom, nom, date_debut, date_fin}* (r7)

Optimisation :

La requête utilise uniquement les colonnes nécessaires (titre, nom, jour_emprunt, mois_emprunt, annee_emprunt), ce qui limite la quantité de données à manipuler et améliore les performances. De plus, les jointures sont effectuées sur des attributs clés (id_emprunt, id_livre, id_adherent), qui sont normalement déjà indexé. Pour d'optimiser le filtrage des emprunts, il serait pertinent d'indexer la colonne statut_emprunt, car elle est utilisée dans la clause WHERE.

Question 2: *Quels livres ont été commandés par des adhérents mais jamais empruntés par personne?*

SQL :

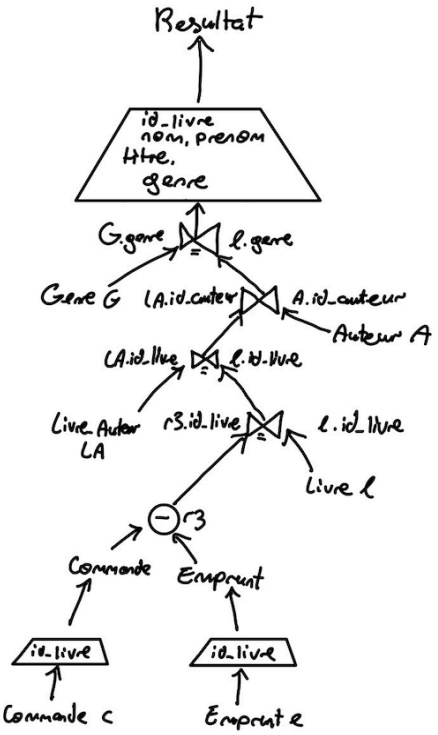
```
SELECT l.id_livre, l.titre,
       CONCAT(a.prenom, ' ', a.nom) AS auteurs,
       g.genre AS nom_genre
FROM Livre l
JOIN Livre_Auteur la ON l.id_livre = la.id_livre
JOIN Auteur a ON la.id_auteur = a.id_auteur
JOIN Genre g ON l.genre = g.genre
WHERE l.id_livre IN (
    SELECT c.id_livre
    FROM Commande c
    WHERE c.id_livre NOT IN (
        SELECT e.id_livre
        FROM Emprunt e
    )
);
```

Requête en algèbre relationnelle :

r1 ← π_{id_livre}(Commande)
r2 ← π_{id_livre}(Emprunt)
r3 ← r1 - r2
r4 ← Livre ⋈_{Livre.id_livre = r3.id_livre} r3
r5 ← r4 ⋈_{Livre.id_livre = Livre_Auteur.id_livre} Livre_Auteur
r6 ← r5 ⋈_{Livre_Auteur.id_auteur = Auteur.id_auteur} Auteur

r7 ← r6 ⋈_{Livre.genre = Genre.genre} Genre
Résultat ← π_{id_livre, titre, prénom, nom, genre}(r7)

Optimisation :



Projection anticipée : On réduit les attributs dès que possible en utilisant π (projection) afin de minimiser les données lors des jointures.
Sélection anticipée (dans ce cas, différence d'ensemble) : On soustrait les livres empruntés des livres commandés le plus tôt possible pour réduire la taille de l'ensemble de travail.
Déplacement des jointures vers le haut : On effectue les jointures avec Livre, Auteur et Genre uniquement après avoir déterminé les livres pertinents.

Question 3 : *On souhaite obtenir le nom, prénom, mail des adhérents avec un ratio retard/nombre d'emprunts supérieur à 10%, ainsi que le ratio. C'est-à-dire que 10% des emprunts de l'adhérent sont en retard.*
(explication : on pourrait par la suite définir un seuil d'acceptance des retard)

SQL :

```sql
WITH emprunts_total AS (
    SELECT
        a.id_adherent,
        a.nom,
        a.prenom,
        a.email,
        COUNT(*) AS total_emprunts
    FROM emprunt e
    JOIN adherent a 1..n<->1: ON e.id_adherent = a.id_adherent
    GROUP BY a.id_adherent, a.nom, a.prenom, a.email
),
emprunts_retard AS (
    SELECT
        a.id_adherent,
        COUNT(*) AS nb_retard
    FROM emprunt e
    JOIN adherent a 1..n<->1: ON e.id_adherent = a.id_adherent
    WHERE e.statut_emprunt = 'en retard'
    GROUP BY a.id_adherent
)

SELECT
    t.nom,
    t.prenom,
    t.email,
    r.nb_retard,
    t.total_emprunts,
    ROUND(CAST(r.nb_retard AS numeric) / t.total_emprunts * 100, 2) AS ratio_retard_percent
FROM emprunts_total t
JOIN emprunts_retard r ON t.id_adherent = r.id_adherent
WHERE CAST(r.nb_retard AS float) / t.total_emprunts > 0.05;
```

Requête en algèbre relationnelle :

$$\text{emprunt\_adh} \leftarrow \pi_{id\_adherent,statut\_emprunt}(Emprunt)$$

$$filtre\_statut \leftarrow \sigma_{statut\_emprunt \neq "ENCOURS"}(emprunt\_adh)$$

$$nb\_emp\_tot \leftarrow \rho_{(id\_adherent,nb\_emprunt)}\left(\gamma_{id\_adherent,count(statut\_emprunt)}(filtre\_statut)\right)$$

$$statut\_retard \leftarrow \sigma_{statut\_emprunt = "ENRETARD"}(emprunt\_adh)$$

$$nb\_emp\_ret \leftarrow \rho_{(id\_adherent,nb\_retard)}\left(\gamma_{id\_adherent,count(statut\_emprunt)}(statut\_retard)\right)$$

$$r_1 \leftarrow nb\_emp\_tot \bowtie_{nb\_emp\_tot.id\_adherent = nb\_emp\_ret.id\_adherent} nb\_emp\_ret$$

$$ratio \leftarrow \rho_{(id\_adherent,nbret,nb\_tot,ratio)}\left(\pi_{id\_adherent,nbret,nb\_tot,((nbret/nb\_tot)*100 \rightarrow ratio)}(r_1)\right)$$

$$filtre\_ratio \leftarrow \sigma_{ratio>10}(ratio)$$

$$adherent \leftarrow \pi_{id\_adherent,nom,prenom,mail}(Adherent)$$

$$résultat \leftarrow \pi_{nom,prenom,mail,ratio}\left(adherent \bowtie filtre\_ratio\right)$$

Optimisation :

Afin d'optimiser notre requête, nous pouvons utiliser des index sur ***id_adherent*** et ***statut_emprunt*** pour accélérer les jointures et les filtrages. Surement, également réduire le nombre de colonnes projetées au strict nécessaire dès le début et peut-être fusionner les *WITH* si possible pour réduire la lecture multiple des tables. Nous pourrions ajouter une condition de filtre (total_emprunts > 0) plus tôt pour limiter le volume de données traité et même (total_emprunts > 2) pour donner au moins deux chances à l'adhérent avant d'interdire les emprunts.

Question 4 : Voir les numéros d'exemplaire, le titre des livres en retard de plus qu'une semaine empruntée par un utilisateur ainsi que son id.

SQL :

```
SELECT E.no_exemplaire, L.titre, E.id_adherent
FROM Emprunt E
JOIN Livre L ON E.id_livre = L.id_livre
WHERE E.statut_emprunt = 'en_cours'
  AND E.date_debut <= CURRENT_DATE - INTERVAL '21 days';
```

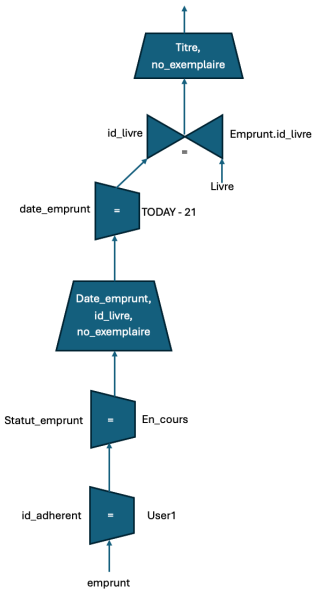
Requête en algèbre relationnelle :

$r_1 \leftarrow \sigma_{\text{id\_adherent AND statut\_emprunt='en\_cours'}}\ \pi_{\text{date\_d'emprunt, id\_livre, no\_exemplaire}} Emprunt$

$r_2 \leftarrow \sigma_{\text{date\_debut <= TODAY - 21}}\ (r_1)$

$r_3 \leftarrow r_2 \bowtie_{r2.\text{id\_livre = Emprunt.id\_livre}}\ (\pi_{\text{titre}}\ Livre)$

Optimisation :



6. DÉVELOPPEMENT

Pour le développement de notre application, nous avons adopté une architecture full-stack moderne. Le **frontend** est développé avec le **framework React**, couplé à **Vite** pour un environnement de développement rapide et efficace. Cette combinaison permet de bénéficier de temps de compilation réduits et d'un rafraîchissement à chaud fluide, améliorant ainsi la productivité des développeurs.

Le **backend** est conçu en **Python 3** en utilisant le framework **FastAPI**, qui permet la création d'API REST performantes et faciles à maintenir. Nous utilisons notamment les éléments suivants :

```
from typing import Union
from fastapi import FastAPI, HTTPException, Depends
from pydantic import BaseModel
from typing import List, Annotated
import models
from database import engine, SessionLocal
from sqlalchemy.orm import Session
```

La **base de données** est gérée par **PostgreSQL**, avec une instance contenue dans **Docker**, facilitant ainsi la gestion de l'environnement et la portabilité du projet. SQLAlchemy est utilisé comme ORM pour gérer les interactions entre les modèles de données et la base de données.

7. BONUS