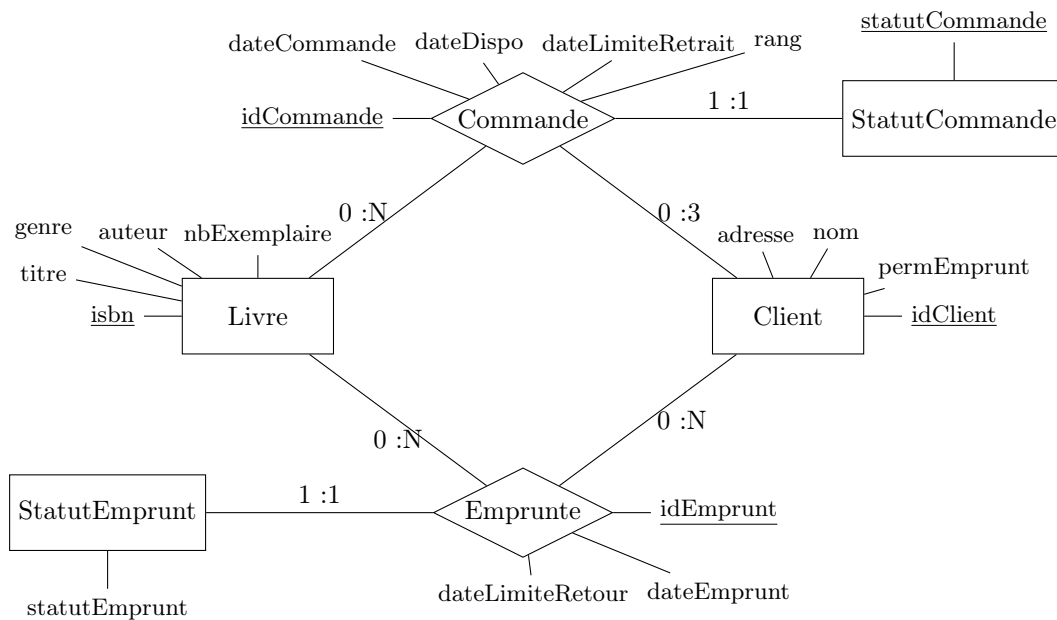


Projet IFT2935

Olivier Larue (20269986)
Claudéric DeRoy (20102789)
Alexandre Toutant (20028191)
Noah Tremblay Taillon (20190661)

22 avril 2025

1 Modélisation en modèle E/A



Explications :

- **isbn et ids** : Ajoutés afin d'identifier facilement les différents éléments.
- **Livre.nbExemplaire** : Représente le nombre d'exemplaires disponibles pour être emprunté. Par exemple, disons que la bibliothèque possède deux exemplaires du *Multidictionnaire de la langue française*, et qu'un client à une commande ou emprunte directement ce livre, le nombre d'exemplaires disponibles sera diminué. Ceci explique aussi la cardinalité de la relation entre **Livre** et **Emprunte**, puisqu'un livre avec plusieurs exemplaires peut donc être emprunté plusieurs fois en même temps.
- **Commande.rang** : Représente le rang de cette commande parmi toutes les commandes sur le même livre. Disons que deux clients créent une commande pour le *Multidictionnaire de la langue française* alors qu'il n'y a qu'un seul exemplaire dans la bibliothèque, les commandes seront traitées selon la politique "première arrivée, première servie" (*FIFO*). Le rang permet de savoir combien de commandes sont devant la notre.
- **Commande.dateCommande** : Représente la date où la commande fut émise, dans le but d'ordonner la file d'attente.

- **Commande.dateDispo** et **Commande.dateLimiteRetrait** : Utilent pour avoir un interval où le livre est réservé au client ayant fait cette commande. Ainsi, lorsque le livre commandé est disponible, ces deux attributs sont mis à jour. Si la date de limite de retrait est dépassée, alors la commande devient expirée et la priorité est donnée à la prochaine commande.
- **StatutCommande.statutCommande** : Permet de faire l'énumération des statuts possibles pour une commande. Les statuts possibles pour une commande sont les suivants :
 - En attente : La commande est passée, mais le client n'a pas encore la priorité sur ce document.
 - Disponible pour le client : Le client à la priorité pour réserver ce document, pour une fenêtre de deux jours.
 - Expirée : Le client avait la priorité pour réserver ce document, mais celui-ci a trop attendu. Il perd donc sa priorité et la commande n'est plus active.
 - Annulée : Le client a décidé d'annuler la commande de son côté ; la commande n'est plus active.
 - Honorée : Le client a réservé ce document. La commande n'est plus active.
- **Client.permEmprunt** : Permet de vérifier si un client peut emprunter ou non. Par exemple, nous pourrions décider qu'un client ayant deux retards actifs serait interdit d'emprunter un troisième document jusqu'au règlement du retard.
- **Client.adresse** : Afin de sauver de l'espace, nous n'avons pas représenté l'adresse comme étant un attribut complexe. Par contre, nous le décomposeront dans le modèle relationnel.
- **StatutEmprunt.statutEmprunt** : Permet de faire l'énumération des statuts possibles pour un emprunt. Ceux-ci permettent de faire des statistiques sur les pratiques des clients. Les statuts possibles pour un emprunt sont les suivants :
 - Retourné : Un emprunt est terminé, sans retard ; celui-ci n'est plus actif et le livre redevient disponible.
 - Régulé : Un emprunt était en retard, mais le client a réglé ce problème. L'emprunt n'est plus actif et le livre redevient disponible.
 - En retard : Un emprunt est en retard ; une action du client est nécessaire. Le livre est encore en possession du client.
 - En cours : Un emprunt est en cours, sans retard. Le livre est encore en possession du client.

2 Transformation en modèle relationnel

Explications :

- Pour commencer, toutes les entités faisant parties du modèle E/A seront conservées comme des tables différentes dans le modèle relationnel.
- Puisque nous n'avons pas de relation 2D 1 : 1, les associations présentes dans le modèle E/A seront aussi présentes dans le modèle relationnel, tout en ayant des clés étrangères pour lier les entités concernées.
- Afin de permettre d'avoir un client avec plusieurs adresses, et qu'une adresse puisse être partagée par plusieurs clients, nous avons créé les tables **AdresseClient** et **Adresse**.
- Afin de nous faciliter la vie plus tard avec la gestion des rangs des commandes, nous avons créé la table **RangCommande**.

Voici le modèle relationnel correspondant :

- Client(idClient, permEmprunt, nom)
- AdresseClient(#idClient, #idAdresse)
- Adresse(idAdresse, rue, ville, code, pays)
- Livre(isbn, titre, genre, auteur, nbExemplaire)
- StatutEmprunt(statutEmprunt)
- StatutCommande(statutCommande)
- Commande(idCommande, #isbn, #idClient, #statutCommande, dateCommande, dateDispo, dateLimiteRetrait)
- RangCommande(#idCommande, #isbn, rang)
- Emprunte(idEmprunt, #isbn, #idClient, #statutEmprunt, dateEmprunt, dateLimiteRetour)

3 Normalisation

Pour la normalisation, nous normaliserons en 3NF. Pour les dépendances fonctionnelles, nous laissons de côté les dépendances triviales.

3.1 Client(idClient, permEmprunt, nom)

3.1.1 Dépendances fonctionnelles

Chaque client a un nom et potentiellement le droit d'emprunter des livres. Donc, l'identifiant client permet de déterminer les deux autres.

$$idClient \rightarrow permEmprunt, nom$$

3.1.2 3NF

Comme nous avons déjà séparé **adresse** en d'autres relation, cette relation est en 1NF ; tous les attributs contiennent une valeur atomique (1NF). Tout attribut non clé ne dépend pas d'une partie de la clé, donc cette relation est 2NF. Tout attribut non clé dépend de la clé, donc cette relation est 3NF.

3.2 AdresseClient(#idClient, #idAdresse)

3.2.1 Dépendances fonctionnelles

Aucune dépendance fonctionnelle non triviale ici, puisque qu'il n'y a pas d'attribut non-clé.

3.2.2 3NF

Les attributs sont atomiques (1NF), ne dépendent pas d'une partie de la clé (2NF), et dépendent directement de la clé (3NF).

3.3 Adresse(idAdresse, rue, ville, code, pays)

3.3.1 Dépendances fonctionnelles

Chaque adresse a un identifiant unique permettant de retrouver ses détails. Donc, cet identifiant permet de retrouver de manière unique les autres attributs.

$$idAdresse \rightarrow rue, ville, code, pays$$

3.3.2 3NF

Les attributs sont atomiques (1NF), ne dépendent pas d'une partie de la clé (2NF), et dépendent directement de la clé (3NF).

3.4 Livre(isbn, titre, genre, auteur, nbExemplaire)

3.4.1 Dépendances fonctionnelles

Chaque livre a un identifiant unique. Plusieurs titres semblables peuvent exister, et un auteur peut avoir écrit plusieurs livres. Donc, seul l'identifiant unique d'un livre permet de déterminer les autres attributs.

$$isbn \rightarrow titre, genre, auteur, nbExemplaire$$

3.4.2 3NF

Les attributs sont atomiques (1NF), ne dépendent pas d'une partie de la clé (2NF), et dépendent directement de la clé (3NF).

3.5 StatutEmprunt(statutEmprunt) et StatutCommande(statutCommande)

3.5.1 Dépendances fonctionnelles

Ces deux relations n'ont qu'un seul attribut, donc aucune dépendance fonctionnelle non triviale.

3.5.2 3NF

Les attributs sont atomiques (1NF), ne dépendent pas d'une partie de la clé (2NF), et dépendent directement de la clé (3NF).

3.6 Commande(idCommande, #isbn, #idClient, #statutCommande, dateCommande, dateDispo, dateLimiteRetrait)

3.6.1 Dépendances fonctionnelles

L'identifiant unique permet de déterminer de manière unique les autres attributs. Rien ne dépend de l'isbn, car il peut y avoir plusieurs commandes sur un même livre ; un client peut faire jusqu'à trois commandes, donc il peut y avoir plusieurs commandes pour un même client ; il peut y avoir plusieurs commandes pour une même date.

$$idCommande \rightarrow isbn, idClient, statutCommande, dateCommande, dateDispo, dateLimiteRetrait$$

3.6.2 3NF

Les attributs sont atomiques (1NF), ne dépendent pas d'une partie de la clé (2NF), et dépendent directement de la clé (3NF).

3.7 RangCommande(#idCommande, #isbn, rang)

3.7.1 Dépendances fonctionnelles

Puisqu'il peut y avoir plusieurs commandes avec un même rang (mais différents livre), et qu'un livre peut avoir plusieurs commandes en attente, seul l'identifiant de commande permet de trouver les deux autres attributs.

$$idCommande \rightarrow isbn, rang$$

3.7.2 3NF

Les attributs sont atomiques (1NF), ne dépendent pas d'une partie de la clé (2NF), et dépendent directement de la clé (3NF).

3.8 Emprunte(idEmprunt, #isbn, #idClient, #statutEmprunt, dateEmprunt, dateLimiteRetour)

3.8.1 Dépendances fonctionnelles

Puisqu'il peut y avoir plusieurs exemplaires d'un même livre, l'isbn ne permet pas de déterminer un emprunt unique. Un client peut avoir plusieurs emprunts (complétés ou actifs), et plusieurs emprunts peuvent avoir le même statut ; même chose pour les dates. Donc, seul l'identifiant d'emprunt permet de déterminer les autres attributs.

$$idEmprunt \rightarrow isbn, idClient, statutEmprunt, dateEmprunt, dateLimiteRetour$$

3.8.2 3NF

Les attributs sont atomiques (1NF), ne dépendent pas d'une partie de la clé (2NF), et dépendent directement de la clé (3NF).

3.9 Schéma final

Le schéma n'a pas changé, puisque toutes les relations étaient déjà 3NF.

- Client(idClient, permEmprunt, nom)
- AdresseClient(#idClient, #idAdresse)
- Adresse(idAdresse, rue, ville, code, pays)
- Livre(isbn, titre, genre, auteur, nbExemplaire)
- StatutEmprunt(statutEmprunt)
- StatutCommande(statutCommande)
- Commande(idCommande, #isbn, #idClient, #statutCommande, dateCommande, dateDispo, dateLimiteRetrait)
- RangCommande(#idCommande, #isbn, rang)
- Emprunte(idEmprunt, #isbn, #idClient, #statutEmprunt, dateEmprunt, dateLimiteRetour)

4 Implémentation

4.1 Schéma des tables

```
1  -- Table Client
2  CREATE TABLE Client (
3      idClient SERIAL PRIMARY KEY,
4      permEmprunt VARCHAR(50),
5      nom VARCHAR(100)
6  );
7
8  -- Table Adresse
9  CREATE TABLE Adresse (
10     idAdresse SERIAL PRIMARY KEY,
11     rue VARCHAR(255),
12     ville VARCHAR(100),
13     code VARCHAR(20),
14     pays VARCHAR(100)
15 );
16
17 -- Table AdresseClient
18 CREATE TABLE AdresseClient (
19     idClient INT,
20     idAdresse INT,
21     PRIMARY KEY (idClient, idAdresse),
22     FOREIGN KEY (idClient) REFERENCES Client(idClient) ON DELETE CASCADE,
23     FOREIGN KEY (idAdresse) REFERENCES Adresse(idAdresse) ON DELETE CASCADE
24 );
25
26 -- Table Livre
27 CREATE TABLE Livre (
28     isbn VARCHAR(20) PRIMARY KEY,
29     titre VARCHAR(255),
30     genre VARCHAR(50),
31     auteur VARCHAR(100),
32     nbExemplaire INT
33 );
34
35 -- Table StatutEmprunt
36 CREATE TABLE StatutEmprunt (
37     statutEmprunt VARCHAR(50) PRIMARY KEY
38 );
39
40 -- Table StatutCommande
41 CREATE TABLE StatutCommande (
42     statutCommande VARCHAR(50) PRIMARY KEY
43 );
44
45 -- Table Commande
```

```

46 CREATE TABLE Commande (
47     idCommande SERIAL PRIMARY KEY,
48     isbn VARCHAR(20),
49     idClient INT,
50     statutCommande VARCHAR(50),
51     dateCommande DATE,
52     dateDispo DATE,
53     dateLimiteRetrait DATE,
54     FOREIGN KEY (isbn) REFERENCES Livre(isbn),
55     FOREIGN KEY (idClient) REFERENCES Client(idClient),
56     FOREIGN KEY (statutCommande) REFERENCES StatutCommande(statutCommande)
57 );
58
59 -- Table RangCommande
60 CREATE TABLE RangCommande (
61     idCommande INT,
62     isbn VARCHAR(20),
63     rang INT,
64     PRIMARY KEY (idCommande, isbn),
65     FOREIGN KEY (idCommande) REFERENCES Commande(idCommande) ON DELETE CASCADE,
66     FOREIGN KEY (isbn) REFERENCES Livre(isbn)
67 );
68
69 -- Table Emprunte
70 CREATE TABLE Emprunte (
71     idEmprunt SERIAL PRIMARY KEY,
72     isbn VARCHAR(20),
73     idClient INT,
74     statutEmprunt VARCHAR(50),
75     dateEmprunt DATE,
76     dateLimiteRetour DATE,
77     FOREIGN KEY (isbn) REFERENCES Livre(isbn),
78     FOREIGN KEY (idClient) REFERENCES Client(idClient),
79     FOREIGN KEY (statutEmprunt) REFERENCES StatutEmprunt(statutEmprunt)
80 );

```

4.2 Population des tables

```

1  -- Insertion dans Client
2  INSERT INTO Client (idClient, permEmprunt, nom) VALUES
3  (1, 'PERM0001', 'Client_1'),
4  (2, 'PERM0002', 'Client_2'),
5  (3, 'PERM0003', 'Client_3'),
6  (4, 'PERM0004', 'Client_4'),
7  (5, 'PERM0005', 'Client_5'),
8  (6, 'PERM0006', 'Client_6'),
9  (7, 'PERM0007', 'Client_7'),
10 (8, 'PERM0008', 'Client_8'),
11 (9, 'PERM0009', 'Client_9'),
12 (10, 'PERM0010', 'Client_10');
13
14 -- Insertion dans Adresse
15 INSERT INTO Adresse (idAdresse, rue, ville, code, pays) VALUES
16 (1, '1 rue Exemple', 'Ville_1', 'H0001', 'Canada'),
17 (2, '2 rue Exemple', 'Ville_2', 'H0002', 'Canada'),
18 (3, '3 rue Exemple', 'Ville_3', 'H0003', 'Canada'),
19 (4, '4 rue Exemple', 'Ville_4', 'H0004', 'Canada'),
20 (5, '5 rue Exemple', 'Ville_0', 'H0005', 'Canada'),
21 (6, '6 rue Exemple', 'Ville_1', 'H0006', 'Canada'),
22 (7, '7 rue Exemple', 'Ville_2', 'H0007', 'Canada'),
23 (8, '8 rue Exemple', 'Ville_3', 'H0008', 'Canada'),
24 (9, '9 rue Exemple', 'Ville_4', 'H0009', 'Canada'),
25 (10, '10 rue Exemple', 'Ville_0', 'H0010', 'Canada');
26
27 -- Insertion dans AdresseClient
28 INSERT INTO AdresseClient (idClient, idAdresse) VALUES
29 (1, 1),

```

```

30 (2, 2),
31 (3, 3),
32 (4, 4),
33 (5, 5),
34 (6, 6),
35 (7, 7),
36 (8, 8),
37 (9, 9),
38 (10, 10);
39
40 -- Insertion dans Livre
41 INSERT INTO Livre (isbn, titre, genre, auteur, nbExemplaire) VALUES
42 ('ISBN0001', 'Titre_1', 'Roman', 'Auteur_1', 6),
43 ('ISBN0002', 'Titre_2', 'Science', 'Auteur_2', 9),
44 ('ISBN0003', 'Titre_3', 'Roman', 'Auteur_3', 2),
45 ('ISBN0004', 'Titre_4', 'Histoire', 'Auteur_4', 7),
46 ('ISBN0005', 'Titre_5', 'Roman', 'Auteur_5', 8),
47 ('ISBN0006', 'Titre_6', 'Histoire', 'Auteur_6', 3),
48 ('ISBN0007', 'Titre_7', 'Science', 'Auteur_7', 6),
49 ('ISBN0008', 'Titre_8', 'Roman', 'Auteur_8', 1),
50 ('ISBN0009', 'Titre_9', 'Science', 'Auteur_9', 4),
51 ('ISBN0010', 'Titre_10', 'Roman', 'Auteur_10', 7);
52
53 -- Insertion dans StatutEmprunt
54 INSERT INTO StatutEmprunt (statutEmprunt) VALUES
55 ('en_cours'),
56 ('retourne'),
57 ('retard');
58
59 -- Insertion dans StatutCommande
60 INSERT INTO StatutCommande (statutCommande) VALUES
61 ('en_attente'),
62 ('disponible'),
63 ('recuperee');
64
65 -- Insertion dans Commande
66 INSERT INTO Commande (idCommande, isbn, idClient, statutCommande, dateCommande, dateDispo,
67     dateLimiteRetrait) VALUES
68 (1, 'ISBN0002', 1, 'recuperee', '2025-04-15', '2025-04-16', '2025-04-22'),
69 (2, 'ISBN0003', 2, 'en_attente', '2025-04-13', '2025-04-15', '2025-04-22'),
70 (3, 'ISBN0004', 3, 'disponible', '2025-04-11', '2025-04-14', '2025-04-22'),
71 (4, 'ISBN0005', 4, 'en_attente', '2025-04-09', '2025-04-13', '2025-04-22'),
72 (5, 'ISBN0006', 5, 'recuperee', '2025-04-07', '2025-04-12', '2025-04-22'),
73 (6, 'ISBN0007', 6, 'disponible', '2025-04-05', '2025-04-11', '2025-04-22'),
74 (7, 'ISBN0008', 7, 'en_attente', '2025-04-03', '2025-04-10', '2025-04-22'),
75 (8, 'ISBN0009', 8, 'recuperee', '2025-04-01', '2025-04-09', '2025-04-22'),
76 (9, 'ISBN0010', 9, 'disponible', '2025-03-30', '2025-04-08', '2025-04-22'),
77 (10, 'ISBN0001', 10, 'en_attente', '2025-03-28', '2025-04-07', '2025-04-22');
78
79 -- Insertion dans RangCommande
80 INSERT INTO RangCommande (idCommande, isbn, rang) VALUES
81 (1, 'ISBN0002', 3),
82 (2, 'ISBN0003', 1),
83 (3, 'ISBN0004', 2),
84 (4, 'ISBN0005', 5),
85 (5, 'ISBN0006', 1),
86 (6, 'ISBN0007', 2),
87 (7, 'ISBN0008', 4),
88 (8, 'ISBN0009', 1),
89 (9, 'ISBN0010', 2),
90 (10, 'ISBN0001', 3);
91
92 -- Insertion dans Emprunte
93 INSERT INTO Emprunte (idEmprunt, isbn, idClient, statutEmprunt, dateEmprunt,
94     dateLimiteRetour) VALUES
95 (1, 'ISBN0002', 1, 'retourne', '2025-03-27', '2025-04-29'),
96 (2, 'ISBN0003', 2, 'en_cours', '2025-03-18', '2025-04-22'),
97 (3, 'ISBN0004', 3, 'retard', '2025-04-12', '2025-05-06'),

```

```

96 (4, 'ISBN0005', 4, 'en_cours', '2025-03-08', '2025-04-22'),
97 (5, 'ISBN0006', 5, 'retourne', '2025-04-07', '2025-05-07'),
98 (6, 'ISBN0007', 6, 'retard', '2025-04-09', '2025-05-01'),
99 (7, 'ISBN0008', 7, 'retourne', '2025-03-18', '2025-05-01'),
100 (8, 'ISBN0009', 8, 'en_cours', '2025-04-05', '2025-05-05'),
101 (9, 'ISBN0010', 9, 'retard', '2025-04-11', '2025-05-01'),
102 (10, 'ISBN0001', 10, 'en_cours', '2025-03-30', '2025-05-03');

```

5 Question/reponse

1. Lister les clients qui ont au moins un emprunt en retard

SQL :

```

1 SELECT e.idClient, c.nom, COUNT(*) AS nbRetards
2 FROM Emprunte e
3 JOIN Client c USING (idClient)
4 WHERE e.statutEmprunt = 'retard'
5 GROUP BY e.idClient, c.nom;

```

Algèbre relationnelle :

$$\pi_{idClient, nom, count(*)}(\sigma_{statutEmprunt='retard'}(Emprunte) \bowtie Client)$$

Optimisation :

- La condition `statutEmprunt = 'retard'` est appliquée avant la jointure, ce qui réduit les lignes traitées.
- L'opérateur `JOIN USING` est plus concis lorsque les noms de colonnes sont identiques (`idClient`).
- L'utilisation de `COUNT(*)` évite de devoir vérifier la nullité d'un champ et est plus efficace.
- Aucune clause `HAVING` n'est nécessaire, car la condition est déjà garantie par le `WHERE`.

2. Lister les emprunts actifs par client avec la date limite de retour

SQL :

```

1 SELECT e.idEmprunt, c.nom, l.titre, e.dateEmprunt, e.dateLimiteRetour
2 FROM Emprunte e
3 JOIN Client c USING (idClient)
4 JOIN Livre l USING (isbn)
5 WHERE e.statutEmprunt = 'en_cours'
6 ORDER BY e.dateLimiteRetour;

```

Algèbre relationnelle :

$$\pi_{idEmprunt, nom, titre, dateEmprunt, dateLimiteRetour}(\sigma_{statutEmprunt='en_cours'}(Emprunte \bowtie Client \bowtie Livre))$$

Optimisation :

- Le filtrage des emprunts en cours est appliqué dès le départ.
- Les jointures utilisent l'opérateur `USING` pour plus de clarté et d'efficacité.
- Le tri est effectué uniquement sur une colonne déjà présente, sans transformation ou calcul.

3. Lister les commandes non récupérées à une date donnée

SQL :

```
1 SELECT c.idCommande, cl.nom, l.titre, c.dateCommande, c.dateLimiteRetrait, c.statutCommande
2 FROM Commande c
3 JOIN Client cl USING (idClient)
4 JOIN Livre l USING (isbn)
5 WHERE c.statutCommande IN ('en_attente', 'disponible')
6 AND c.dateLimiteRetrait >= DATE '2025-04-17'
7 ORDER BY c.dateLimiteRetrait;
```

Algèbre relationnelle :

$$\pi_{idCommande, nom, titre, dateCommande, dateLimiteRetrait, statutCommande} \left(\sigma_{statutCommande \in \{en_attente, disponible\} \wedge dateLimiteRetrait \geq 2025-04-17} \left(Commande \bowtie Client \bowtie Livre \right) \right)$$

Optimisation :

- Les lignes sont filtrées en amont selon deux conditions restrictives : le statut et la date limite.
- L’usage explicite du type DATE assure une comparaison efficace et sans conversion implicite.
- La requête ne sélectionne que les colonnes nécessaires à l’affichage.

4. Compter le nombre total de commandes par genre de livre

SQL :

```
1 SELECT l.genre, COUNT(*) AS nbCommandes
2 FROM Commande c
3 JOIN Livre l USING (isbn)
4 GROUP BY l.genre
5 ORDER BY nbCommandes DESC;
```

Algèbre relationnelle :

$$\gamma_{genre, count(*)}(Commande \bowtie Livre)$$

Optimisation :

- Le regroupement est fait uniquement sur le champ **genre**, ce qui simplifie le plan d’exécution.
- L’utilisation de COUNT(*) est plus performante pour le comptage global.
- L’ordre décroissant est appliqué sur une agrégation déjà calculée, sans besoin de calcul supplémentaire.

6 Développement

Le projet a été conçu comme une application full stack. Le front-end a été développé avec React, un framework JavaScript qui facilite le développement d’interfaces. Pour le back-end, nous avons utilisé Express.js, un framework léger et rapide à prendre en main.

Le tout est hébergé dans un répertoire GitHub, incluant le front-end, le back-end, ainsi que les scripts nécessaires à la base de données. Nous avons choisi de tout centraliser dans un seul dépôt étant donné la petite taille du projet, ce qui simplifie la gestion et la collaboration.

Nous avons utilisé Sequelize comme ORM pour faciliter les requêtes entre Express.js et la base de données PostgreSQL. Sequelize s’intègre bien à l’environnement JavaScript, ce qui a renforcé notre choix.

La base de données est gérée dans un conteneur Docker, via un script Docker Compose. Ce dernier configure automatiquement un serveur PostgreSQL local, dont les données sont initialement poplées à l'aide d'un fichier seed.sql, permettant à chaque utilisateur d'avoir les mêmes données initiales.

Pour faciliter l'installation et l'exécution du projet, un fichier README.md détaille les étapes nécessaires. Le développement a été effectué avec Visual Studio Code, en s'appuyant sur la documentation officielle de chaque technologie.

Par exemple :

- Pour Express.js : expressjs.com/en/starter/installing.html
- Pour React : react.dev/learn/installation et react.dev/learn
- Pour Sequelize : <https://sequelize.org/docs/v6/getting-started/>

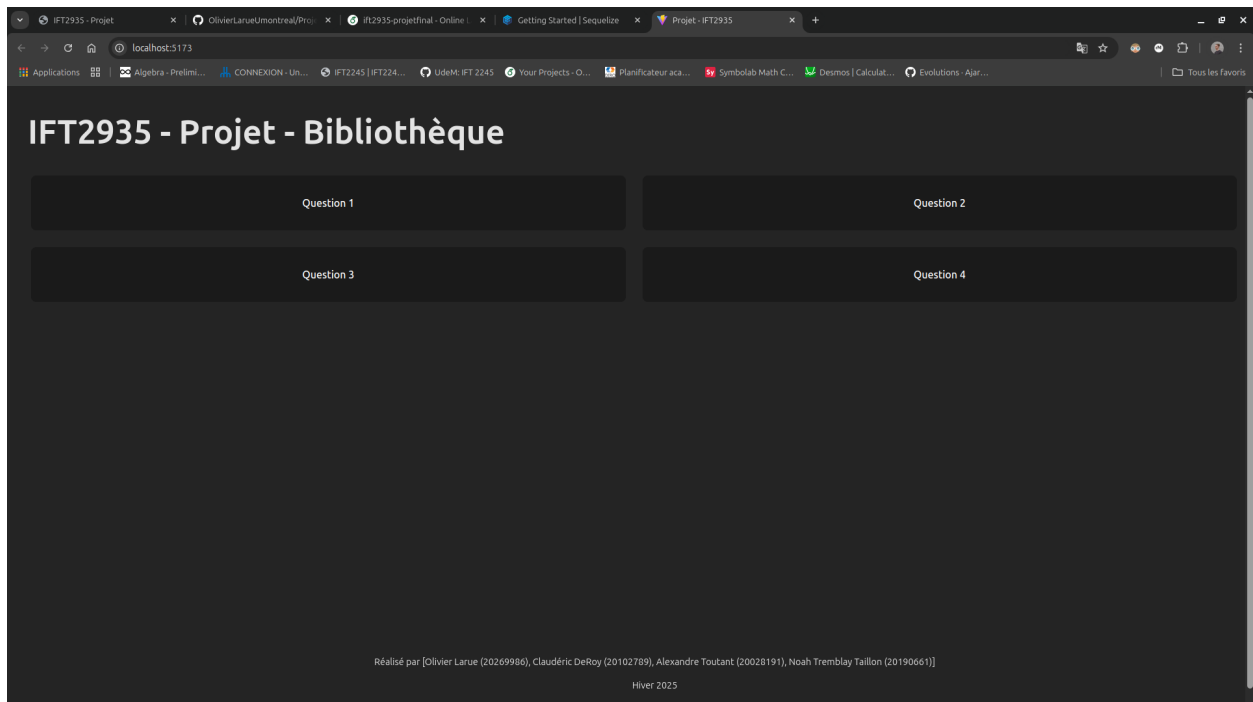
Le projet comporte 5 pages :

- La page d'accueil (HomePage)
- Pages 1-4. Une page pour chaque questions

L'api du backend comporte sur 5 routes :

- `http://localhost:5000/api` (pour verifier si l'api fonctionne. Route de base de l'api.)
- `http://localhost:5000/api/question1` (pour obtenir le contenu en format JSON de la requête pour la question 1)
- `http://localhost:5000/api/question2` (pour obtenir le contenu en format JSON de la requête pour la question 2)
- `http://localhost:5000/api/question3` (pour obtenir le contenu en format JSON de la requête pour la question 3)
- `http://localhost:5000/api/question4` (pour obtenir le contenu en format JSON de la requête pour la question 4)

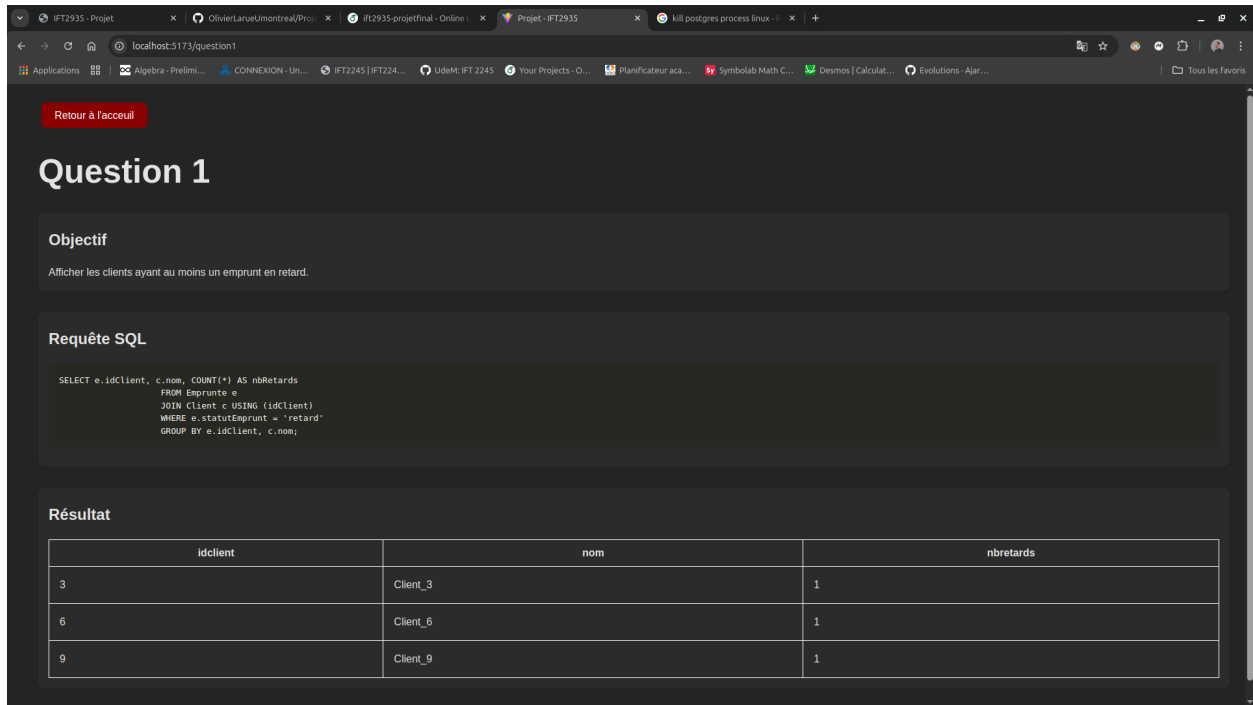
Une fois le projet lancé et que vous accédez à `localhost:5173`, vous arrivez sur la page d'accueil de l'application :



La page d'accueil contient quatre boutons, chacun représentant une question que nous avons formulée précédemment. Lorsqu'on clique sur l'un de ces boutons, la page affiche :

- la question sélectionnée,
- la requête SQL correspondante,
- le résultat de cette requête.

Par exemple, pour la question 1, voici ce que l'on obtient :



The screenshot shows a web browser window with a dark theme. The address bar shows 'localhost:5173/question1'. The page has a red button 'Retour à l'accueil' at the top left. The main content is titled 'Question 1'. Below the title, there is a section 'Objectif' with the text 'Afficher les clients ayant au moins un emprunt en retard.' Below that is a section 'Requête SQL' containing a SQL query. At the bottom is a section 'Résultat' containing a table with three columns: 'idclient', 'nom', and 'nbretards'. The table has three rows of data.

Question 1

Objectif
Afficher les clients ayant au moins un emprunt en retard.

Requête SQL

```
SELECT e.idClient, c.nom, COUNT(*) AS nbRetards
FROM Emprunte e
JOIN Client c USING (idClient)
WHERE e.statutEmprunt = 'retard'
GROUP BY e.idClient, c.nom;
```

Résultat

idclient	nom	nbretards
3	Client_3	1
6	Client_6	1
9	Client_9	1

La même logique s'applique pour les quatre questions : chaque page est dédiée à une question spécifique et affiche la question, la requête SQL utilisée, ainsi que le résultat obtenu.

7 Installation

Une vidéo ainsi que des instructions dans un dépôt GitHub expliquent comment procéder pour l'installation et l'exécution du projet.