Test d'exécution des fonctionnalités du système de gestion de bibliothèque

1. Introduction

Ce document présente les tests effectués sur le **système de gestion de bibliothèque**. L'objectif de ces tests est de **valider le bon fonctionnement des différentes fonctionnalités**, en s'assurant qu'elles répondent aux exigences fonctionnelles définies.

Les tests ont porté sur les fonctionnalités suivantes :

- **Gestion des catégories** : ajout, modification, suppression et consultation des catégories de livres.
- **Gestion des auteurs** : ajout, recherche, suppression et consultation des informations des auteurs.
- **Gestion des livres** : enregistrement, mise à jour, suppression, affichage et recherche des livres dans le système.
- **Gestion des membres** : enregistrement, suppression, recherche des membres, ainsi que consultation des emprunts associés.
- **Gestion des emprunts** : enregistrement, retour, suppression et affichage des emprunts existants.
- **Gestion des pénalités** : affichage des emprunts avec pénalités et consultation des montants respectifs.

Chaque fonctionnalité a été **testée individuellement** afin de garantir que le système fonctionne correctement dans un environnement de production.

---

2. Méthodologie

Les tests ont été réalisés en suivant une **approche systématique et structurée**, basée sur l'interaction avec les **menus interactifs** du système de gestion de bibliothèque.

2.1. Processus de test

Chaque fonctionnalité a été testée **individuellement**, en simulant des opérations réelles effectuées par un utilisateur.

- **Exécution des tests via les menus interactifs** :
    - Les tests ont été réalisés en naviguant directement dans l'interface du programme.
    - Chaque opération (ajout, modification, suppression, recherche, etc.) a été validée via le menu correspondant.
    - Cela a permis d'évaluer le bon fonctionnement du système dans des conditions proches de l'utilisation réelle.
- **Validation des résultats** :
    - Pour chaque test, les **résultats attendus** ont été comparés aux **résultats réels** produits par le système.
    - Exemples :
        - Vérification de l'ajout d'une catégorie et de son affichage dans la liste.
        - Affichage correct des livres enregistrés.
        - Gestion correcte des erreurs et messages d'alerte en cas de saisie incorrecte.
- **Documentation des tests** :
    - Des **captures d'écran** ont été prises à chaque étape clé.
    - Elles illustrent les différentes interactions avec le système, les résultats obtenus et les éventuelles erreurs rencontrées.
- **Suivi et correction des erreurs** :
    - En cas de **comportement inattendu** ou d'**anomalie**, les erreurs ont été notées et documentées via des captures d'écran.
    - Une analyse a été effectuée pour identifier la cause des problèmes et y apporter une correction.
    - Une fois corrigées, les fonctionnalités ont été **testées à nouveau** pour s'assurer que les erreurs ne se reproduisaient pas.

3. Tests de fonctionnalité

3.1. Gestion des catégories

Menu testé :

- Ajouter une catégorie
- Modifier une catégorie
- Supprimer une catégorie
- Lister toutes les catégories

- Rechercher des catégories par mot-clé

Capture d'écran 1 : *Ajouter une catégorie*

Capture d'écran 2 : *Modifier une catégorie*
(Capture du processus de modification d'une catégorie.)

Capture d'écran 3 : *Supprimer une catégorie*
(Capture montrant la suppression d'une catégorie.)

Capture d'écran 4 : Lister toutes les catégories

(Capture montrant l'affichage des catégories.)

Capture d'écran 5 : Rechercher des catégories par mot-clé

3.2. Gestion des auteurs

Menu testé :

- Ajouter un auteur
- Lister tous les auteurs
- Supprimer un auteur
- Rechercher un auteur par email
- Rechercher un auteur par ID

Capture d'écran 1 : *Ajouter un auteur*
(Capture d'écran de l'ajout d'un auteur.)

Capture d'écran 2 : *Rechercher un auteur par email*
(Capture montrant la recherche d'un auteur via l'email.)

Capture d'écran 3 : *Rechercher un auteur par email*
(Capture montrant la recherche d'un auteur via l'email.)

Capture d'écran 4 : *Supprimer un auteur*
(Capture montrant la suppression d' un auteur.)

Capture d'écran 5 : *Rechercher un auteur par ID*
(Capture montrant la recherche d'un auteur via l'IDl.)

3.3. Gestion des livres

Menu testé :

- Ajouter un livre
- Afficher un livre
- Afficher tous les livres
- Mettre à jour un livre
- Rechercher un livre par catégorie
- Supprimer un livre

Capture d'écran 1 : *Ajouter un livre*
(Capture montrant l'ajout d'un livre dans le système.)

Capture d'écran 2 : Afficher un livre
(Capture montrant l'affichage d'un livre dans le système.)

Capture d'écran 3 : Afficher tous les livres
(Capture montrant l'affichage de tous les livres dans le système.)

Capture d'écran 4 : Mettre à jour un livre
(Capture montrant la mise à jour d'un livre dans le système.)

Capture d'écran 5 : Rechercher un livre par catégorie
(Capture montrant la recherche d'un livre par catégorie dans le système.)

Capture d'écran 6 : Supprimer un livre
(Capture montrant la suppression d'un livre par catégorie dans le système.)

3.4. Gestion des membres

Menu testé :

- Enregistrer un membre
- Supprimer un membre
- Rechercher les membres par mot-clé
- Rechercher un membre par ID
- Afficher les emprunts d'un membre
- Empêcher un doublon d’email

Capture d'écran 1 : *Enregistrer un membre*
(Capture d'écran montrant l'enregistrement d'un nouveau membre.)

Capture d'écran 2 : Supprimer un membre
(Capture d'écran montrant la suppression d'un membre.)

Capture d'écran 3 : Rechercher les membres par mot-clé
(Capture d'écran montrant la recherche des membres.)

Capture d'écran 4 : *Afficher un membre*
(Capture d'écran montrant l'affichage d'un membre.)

Capture d'écran 5 : *Afficher les emprunts d'un membre*
(Capture montrant les emprunts d'un membre.)

Capture d'écran 6 : *Empêcher un doublon d'email*
(Capture montrant la non duplication d'email.)

3.5. Gestion des emprunts

Menu testé :

- Enregistrer un emprunt
- Retourner un emprunt
- Afficher tous les emprunts
- Supprimer un emprunt
- Afficher la liste des emprunts

Capture d'écran 1 : *Enregistrer un emprunt*
(Capture montrant le processus d'enregistrement d'un emprunt.)

Capture d'écran 2 : *Afficher tous les emprunts*
(Capture montrant le processus d'affichage de tous les emprunts.)

Capture d'écran 3 : Supprimer un emprunt
(Capture montrant le processus de suppression d'un emprunt.)

Capture d'écran 4 : *Retourner un emprunt*
(Capture montrant le retour d'un emprunt.)

Capture d'écran 4 : *Afficher la liste des pénalités*
(Capture montrant la liste des pénalités.)

4. Résultats et observations

Gestion des catégories

📌 Gestion des catégories

1️⃣Problème : Validation trop restrictive des noms de catégorie

- **Symptôme** : L'expression régulière initiale (^[a-zA-Z]+$) n'autorisait que les lettres, empêchant des noms comme :
  - Science-Fiction (cause : tiret)
  - Développement Personnel (cause : espace)
  - Informatique & Programmation (cause : &)
- **Solution** :
  ✅ Passage à une regex plus souple : ^[a-zA-Z&\s-]+$, mais encore quelques erreurs avec les accents.
  ✅ Correction finale avec ^[\p{L}&\s-]+$, acceptant :
  - Lettres avec accents (Développement)
  - Espaces (Développement Personnel)
  - Tirets (Science-Fiction)
  - & (Informatique & Programmation)

2️⃣Problème : Ajout de catégories existantes déclenche une erreur critique

- **Symptôme** : Lorsqu'un utilisateur essayait d'ajouter une catégorie déjà existante, une erreur bloquante était affichée dans les logs.
- **Solution** :
  ✅ Capture correcte de CategoryAlreadyExistsException dans CategoryController.
  ✅ Remplacement du message d'erreur critique (Logger.logError) par un simple avertissement (Logger.logWarn).

3️⃣Problème : Erreurs de connexion à PostgreSQL lors du renommage de la base

- **Symptôme** : Après le passage de bibli à library_management, plusieurs erreurs sont apparues :
  - FATAL: authentification par mot de passe échouée pour l'utilisateur "your_user"
  - ERROR: La base de données 'library_management' n'existe pas.

- **Solution** :
  ✅ Ajout d'un mécanisme pour vérifier et créer automatiquement la base si elle n'existe pas.
  ✅ Correction des paramètres de connexion et test de reconnection automatique.

📌 **Conclusion**

🔹 Les principales difficultés concernaient :
1️⃣ La validation des noms de catégories
2️⃣ La gestion des doublons lors de l'ajout
3️⃣ La connexion et création automatique de la base PostgreSQL

📌 Gestion des auteurs

1️⃣ **Problème : Validation des noms d'auteur (prénom et nom)**

- **Symptôme** : Les prénoms et noms d'auteurs ne prenaient pas en compte les accents et certains caractères spéciaux.
- **Solution** :
  ✅ Mise à jour du regex en `^[\p{L}&\s-]+$` pour accepter :
  - Lettres avec accents (Émile Zola)
  - Espaces (Jean Paul Sartre)
  - Tirets (Jean-Pierre Jeunet)
  - & (L. & A. de Vinci)

2️⃣ **Problème : Mauvaise gestion des emails existants**

- **Symptôme** : Lors de la création d'un auteur, si l'email était déjà utilisé, l'application levait une exception sans message clair.
- **Solution** :
  ✅ Capture correcte de `AuthorEmailAlreadyExistsException` dans `AuthorController` avec un message clair via `Logger.logWarn()`.

3️⃣ **Problème : Erreur `InputMismatchException` lors de la saisie d'un ID**

- **Symptôme** : Si l'utilisateur entrait un ID non numérique (ex. une lettre), l'application levait une exception et plantait.

- **Solution** :
  ✅ Ajout d'une validation (scanner.hasNextInt()) avant de lire l'ID pour éviter les erreurs.

📌 **Conclusion**

🔹 Les principales difficultés concernaient :
1️⃣La validation des noms d'auteurs
2️⃣La gestion des emails en double
3️⃣La saisie d'un ID non valide

📌 Gestion des livres

1️⃣Problème : Recherche par catégorie sensible à la casse

- **Symptôme** : Rechercher "histoire" ne trouvait pas "Histoire", car la recherche était sensible à la casse.
- **Solution** :
  ✅ Modification de la requête SQL en LOWER(c.category_name) = LOWER(?) pour ignorer la casse.

2️⃣Problème : Erreur NullPointerException lors de la recherche par catégorie

- **Symptôme** : Si la catégorie n'existait pas, une NullPointerException était levée.
- **Solution** :
  ✅ Ajout d'un contrôle if (books == null || books.isEmpty()) pour éviter l'erreur.

3️⃣Problème : Affichage non lisible des résultats de recherche

- **Symptôme** : Les livres trouvés étaient affichés en vrac, rendant la lecture difficile.
- **Solution** :
  ✅ Utilisation de System.out.printf() pour un affichage aligné et lisible.

4️⃣Problème : InputMismatchException lors de la mise à jour d'un livre

- **Symptôme** : Entrer une donnée invalide (ex. un texte au lieu d'un entier) lors de la mise à jour d'un livre levait une exception.
- **Solution** :
  ✅ Vérification avec scanner.hasNextInt() avant lecture pour éviter l'erreur.

5️⃣Problème : Mise à jour échouée si l'ID de l'auteur ou de la catégorie n'existe pas

- **Symptôme** : Si un ID d'auteur ou de catégorie invalide était fourni, la mise à jour échouait sans message clair.
- **Solution** :
  ✅ Vérification préalable de l'existence des IDs avant d'appliquer la mise à jour.

📌 Conclusion

🔹 Les principales difficultés concernaient :
1️⃣La recherche par catégorie insensible à la casse
2️⃣La gestion des erreurs lors de la recherche
3️⃣L'affichage des résultats de recherche
4️⃣La saisie d'une valeur incorrecte lors de la mise à jour
5️⃣La vérification des IDs avant mise à jour

Gestion des membres

1️⃣Problème : Email en double non géré correctement

- **Symptôme** : Lorsqu'un membre tentait de s'inscrire avec un email déjà utilisé, une exception était levée, mais la vérification de l'existence de l'email n'était pas bien intégrée.
- **Solution** :
  ✅ Ajout de la méthode isEmailTaken() dans MemberDAOImpl pour vérifier l'existence de l'email avant l'insertion.
  ✅ Modification de registerMember() pour appeler isEmailTaken() et afficher un message clair si l'email existe déjà.

2️⃣Problème : Mauvaise gestion des erreurs SQL lors de l'inscription d'un membre

- **Symptôme** : En cas d'erreur SQL (ex. : problème de connexion à la base de données), l'exception n'était pas bien capturée, ce qui entraînait des erreurs inattendues.

- **Solution** :
  ✅ Ajout d'un try-catch plus robuste dans registerMember() pour capturer et loguer les erreurs SQL avec des messages clairs.
  ✅ Ajout d'un return propre en cas d'échec, au lieu de laisser l'application planter.

3️⃣ **Problème : Normalisation des emails pour éviter les doublons dus à la casse**

- **Symptôme** : Deux emails identiques mais avec des majuscules différentes (Alice@example.com et alice@example.com) étaient considérés comme distincts.
- **Solution** :
  ✅ Normalisation des emails en minuscules (email.trim().toLowerCase()) avant l'insertion et la vérification.

4️⃣ **Problème : Absence de validation avancée sur les noms et emails**

- **Symptôme** : L'utilisateur pouvait entrer des caractères invalides dans les champs prénom, nom et email.

Solution :
✅ Ajout de validations robustes dans MemberHandler.registerMember(), avec des regex acceptant les accents, tirets et espaces.
✅ Vérification stricte de l'email avec une regex améliorée :
java
Copier le code
"^[\\p{L}0-9._%+-]+@[\\p{L}0-9.-]+\\.[A-Za-z]{2,10}$"

- ✅ Ajout de messages d'erreur clairs avec Logger.logWarn().

5️⃣ **Problème : La méthode isEmailTaken() n'était pas accessible depuis MemberHandler**

- **Symptôme** : MemberHandler ne pouvait pas vérifier si un email était déjà utilisé.
- **Solution** :
  ✅ Ajout d'un appel à memberService.isEmailTaken(email) avant d'insérer un nouveau membre.
  ✅ Réorganisation de la logique d'inscription pour que MemberHandler puisse déléguer cette vérification à MemberServiceImpl.

📌 Conclusion

🔹 Les principales difficultés concernaient :
1️⃣ La gestion des emails en double
2️⃣ La gestion des erreurs SQL

3️⃣La normalisation des emails
4️⃣La validation avancée des noms et emails
5️⃣L'intégration correcte de la vérification `isEmailTaken()`

3.6. Gestion des pénalités

Menus testés :

- Afficher les emprunts avec pénalités
- Afficher le montant des pénalités

Difficultés rencontrées et solutions apportées

1️⃣Mauvais calcul des pénalités après dépassement du délai

- **Problème** : Les pénalités n'étaient pas recalculées après un retour tardif.
- **Solution** : Ajout d'un recalcul automatique des pénalités avant chaque affichage.

2️⃣Affichage confus du montant des pénalités

- **Problème** : Les montants étaient affichés sans unité (`5` au lieu de `5€`).
- **Solution** : Ajout de `System.out.printf("%.2f €")` pour un affichage lisible.

Captures d'écran :

📸 Capture d'écran 1 : Afficher les emprunts avec pénalités

5. Conclusion

Synthèse des fonctionnalités testées

L'ensemble des tests réalisés a permis d'évaluer les principales fonctionnalités du système de gestion de bibliothèque. Ces tests ont couvert plusieurs aspects critiques, notamment :

- **Gestion des catégories** : Vérification de l'ajout, de la modification, de la suppression et de la consultation des catégories de livres.
- **Gestion des auteurs** : Validation de l'enregistrement des auteurs, de la gestion des doublons d'email, de la recherche et de la suppression d'un auteur.
- **Gestion des livres** : Contrôle de l'enregistrement, de la mise à jour, de l'affichage et de la suppression des livres, ainsi que de leur association avec des auteurs et des catégories.

- **Gestion des membres** : Évaluation du processus d'inscription des membres, de la suppression et de la recherche des membres dans le système.
- **Gestion des emprunts** : Analyse du processus d'emprunt et de retour des livres, impactant la mise à jour du nombre d'exemplaires disponibles et la gestion des pénalités en cas de retard.
- **Exploitation des données** : Vérification des requêtes SQL permettant la récupération et l'affichage des informations relatives aux livres, aux emprunts en cours et aux pénalités applicables.

Impact des tests sur la validation du système

Les tests effectués ont permis de valider le bon fonctionnement du système et d'identifier plusieurs anomalies qui ont été corrigées. Les principaux apports de cette phase de test sont :

- **Robustesse et cohérence des données** :
    - Vérification de l'intégrité des relations entre les différentes entités de la base de données (livres, auteurs, membres, emprunts, catégories).
    - Mise en place de contraintes d'unicité et de validation des entrées pour prévenir les incohérences.
- **Fiabilité des opérations métier** :
    - Simulation des cas d'utilisation standards et exceptionnels pour garantir le respect des règles de gestion (ex. : interdiction d'un emprunt si aucun exemplaire n'est disponible, gestion des retours et des pénalités).
    - Optimisation des requêtes SQL pour assurer une récupération efficace des données.
- **Amélioration de la gestion des erreurs** :
    - Identification et correction des erreurs potentielles (ex. : saisie invalide d'un ID, gestion des cas où un auteur ou une catégorie n'existe pas).
    - Ajout de messages d'erreur explicites pour faciliter l'interaction avec l'utilisateur.

Évaluation de la préparation à la production

Les tests réalisés démontrent que le système est **opérationnel et conforme aux exigences fonctionnelles définies**. Il est en mesure de gérer efficacement les principales opérations attendues d'un système de gestion de bibliothèque. Toutefois, pour garantir une transition optimale vers un environnement de production, les actions suivantes sont recommandées :

- **Tests utilisateurs** : Mise en place d'une phase de validation en conditions réelles avec des utilisateurs afin d'identifier d'éventuelles améliorations ergonomiques ou fonctionnelles.
- **Optimisation des performances** : Surveillance des temps d'exécution des requêtes sur un volume de données plus conséquent pour identifier d'éventuels goulets d'étranglement.

- **Sécurisation des accès** : Vérification de la gestion des droits d'accès et mise en place d'une authentification si nécessaire pour renforcer la sécurité du système.

En conclusion, le système est prêt pour une **phase de déploiement supervisé** et peut être envisagé pour une mise en production après validation des dernières recommandations. 🚀

6. Annexes

Cette section regroupe quelques requêtes SQL utilisées pour interroger la base de données du système de gestion de bibliothèque. Elles permettent d'extraire des informations essentielles sur les livres, les emprunts et les pénalités.

6.1. Requêtes SQL

📌 Requête pour afficher un livre spécifique

Cette requête permet d'afficher les détails d'un livre, y compris son titre, le nombre d'exemplaires disponibles, son auteur et sa catégorie.

```
SELECT  b.book_id AS id_du_livre,  b.title AS titre_du_livre,  b.number_of_copies AS nombre_de_copies,    CONCAT(a.first_name, ' ', a.last_name) AS nom_auteur,  a.author_email AS email_auteur,    c.category_name AS categorie

FROM   Book b LEFT JOIN   Book_Author ba ON b.book_id = ba.book_id

LEFT JOIN  Author a ON ba.author_id = a.author_id

LEFT JOIN   Books_Category bc ON b.book_id = bc.book_id

LEFT JOIN   Category c ON bc.category_id = c.category_id

WHERE   b.book_id = 1;
```

📌 Requête pour afficher toutes les informations sur les livres

Cette requête liste tous les livres avec leurs auteurs et leurs catégories associées.

```
SELECT   b.book_id,  b.title,   b.number_of_copies, a.first_name,   a.last_name,
```

```
a.author_email,  c.category_name

FROM   Book b LEFT JOIN   Book_Author ba ON b.book_id = ba.book_id

LEFT JOIN  Author a ON ba.author_id = a.author_id

LEFT JOIN    Books_Category bc ON b.book_id = bc.book_id

LEFT JOIN  Category c ON bc.category_id = c.category_id;
```

📌 Requête pour afficher tous les emprunts

Cette requête permet d'afficher tous les emprunts avec les informations des membres, les dates de prêt et de retour, ainsi que les livres empruntés avec leurs auteurs et catégories.

```
SELECT l.loan_id, m.first_name, m.last_name, m.email,l.loanDate,l.dueDate,
l.returnDate, bl.book_id, b.title, b.number_of_copies,

   STRING_AGG(DISTINCT a.first_name || ' ' || a.last_name, ', ') AS authors,

   STRING_AGG(DISTINCT c.category_name, ', ') AS categories  -- Ajout de la virgule
ici

FROM Loan l

JOIN Member m ON l.member_id = m.member_id

JOIN Book_Loan bl ON l.loan_id = bl.loan_id

JOIN Book b ON bl.book_id = b.book_id

LEFT JOIN Book_Author ba ON b.book_id = ba.book_id

LEFT JOIN Author a ON ba.author_id = a.author_id

LEFT JOIN Books_Category bc ON b.book_id = bc.book_id

LEFT JOIN Category c ON bc.category_id = c.category_id

GROUP BY l.loan_id, m.first_name, m.last_name, m.email, l.loanDate, l.dueDate,
l.returnDate, bl.book_id, b.title, b.number_of_copies

ORDER BY l.loanDate DESC;
```

📌 Requête pour afficher les montants des pénalités

Cette requête liste les emprunts en retard avec les informations des membres concernés.

```
SELECT  l.loan_id, l.loanDate,  l.dueDate,  l.returnDate,  l.member_id,  m.first_name,  m.last_name,  m.email FROM  Loan l  JOIN  Member m ON l.member_id = m.member_id  WHERE  l.returnDate IS NULL  AND l.dueDate < CURRENT_TIMESTAMP;
```

📌 Requête pour afficher la liste des membres inscrits

Cette requête permet d'afficher tous les membres avec leur date d'adhésion.

```
SELECT  member_id AS id_du_membre,  first_name AS prenom,  last_name AS nom,  email,  adhesion_date AS date_adhesion FROM  Member ORDER BY  adhesion_date DESC;
```

📌 Requête pour afficher les livres empruntés par un membre donné

Cette requête permet d'afficher tous les livres empruntés par un membre spécifique grâce à son email.

```
SELECT m.member_id,  m.first_name,  m.last_name,  m.email,  b.book_id,  b.title,  l.loanDate,  l.dueDate,  l.returnDate FROM Member m JOIN  Loan l ON m.member_id = l.member_id JOIN  Book_Loan bl ON l.loan_id = bl.loan_id JOIN  Book b ON bl.book_id = b.book_id WHERE  m.email = 'emmanuel.ndjomo@gmail.com' ORDER BY l.loanDate DESC;
```

📌 Requête pour afficher les livres non encore retournés par un membre donné

Cette requête liste les emprunts en cours d'un membre, c'est-à-dire ceux dont returnDate est NULL.

```
SELECT m.member_id, m.first_name, m.last_name, m.email, b.book_id, b.title, l.loanDate, l.dueDate FROM Member m JOIN Loan l ON m.member_id = l.member_id JOIN Book_Loan bl ON l.loan_id = bl.loan_id JOIN Book b ON bl.book_id = b.book_id WHERE m.email = 'emmanuel.ndjomo@gmail.com' -- Remplacer par l'email du membre concerné AND l.returnDate IS NULL ORDER BY l.dueDate ASC;
```

📌 Requête pour afficher la liste des auteurs et leurs livres

Cette requête permet d'afficher tous les auteurs ainsi que les livres qu'ils ont écrits.

```
SELECT a.author_id, CONCAT(a.first_name, ' ', a.last_name) AS nom_auteur, a.author_email, b.book_id, b.title FROM Author a JOIN Book_Author ba ON a.author_id = ba.author_id JOIN Book b ON ba.book_id = b.book_id ORDER BY nom_auteur, b.title;
```

📌 Requête pour afficher les membres ayant des pénalités

Cette requête permet d'afficher les membres qui ont au moins une pénalité, avec les montants cumulés.

```
SELECT m.member_id, m.first_name, m.last_name, m.email, SUM(lp.penalty_amount) AS total_penalites FROM Loan_Penalties lp JOIN Loan l ON lp.loan_id = l.loan_id JOIN Member m ON l.member_id = m.member_id GROUP BY m.member_id, m.first_name, m.last_name, m.email HAVING SUM(lp.penalty_amount) > 0 ORDER BY total_penalites DESC;
```

📌 Requête pour afficher les catégories et le nombre de livres par catégorie

Cette requête liste toutes les catégories et indique combien de livres appartiennent à chaque catégorie.

```
SELECT c.category_id, c.category_name, COUNT(bc.book_id) AS nombre_de_livres FROM Category c LEFT JOIN Books_Category bc ON c.category_id = bc.category_id GROUP BY c.category_id, c.category_name ORDER BY nombre_de_livres DESC;
```

📌 Requête pour afficher les 5 derniers emprunts effectués

Cette requête affiche les cinq derniers emprunts enregistrés dans le système.

```
SELECT l.loan_id, m.first_name, m.last_name, m.email, b.title, l.loanDate,
l.dueDate, l.returnDate FROM Loan l JOIN Member m ON l.member_id = m.member_id
JOIN Book_Loan bl ON l.loan_id = bl.loan_id JOIN Book b ON bl.book_id = b.book_id
ORDER BY l.loanDate DESC LIMIT 5;
```

📌 Requête pour afficher les livres qui n'ont jamais été empruntés

Cette requête liste tous les livres qui n'ont jamais été associés à un prêt dans la table Book_Loan.

```
SELECT b.book_id, b.title, b.number_of_copies FROM Book b LEFT JOIN Book_Loan
bl ON b.book_id = bl.book_id WHERE bl.book_id IS NULL;
```