

## Rapport d'analyse et de validation du projet de gestion de bibliothèque

## Réalisé par:

- ❖ Sabrina Lamghari
- Cyril Rouillon
- ❖ Hamid Amediaz



Le déroulement du projet de base de données pour la bibliothèque a suivi une organisation structurée en plusieurs phases : conception, implémentation, validation et documentation.

La répartition des tâches s'est effectuée comme suit:

- <u>Sabrina:</u> La gestion des emprunts, le suivi des prolongations et l'analyse des comportements des abonnés et de la popularité des ouvrages.
- <u>Hamid:</u> L'optimisation des emprunts, la gestion des réservations et l'intégration des nouvelles bibliothèques au réseau.
- **Cyril:** La gestion des rôles et des privilèges d'accès, l'optimisation des transferts d'ouvrages pour minimiser les coûts logistiques.

Pour regarder ce que chaque membre du groupe a fait, veuillez le trouver sur GitHub dans des fichiers séparés.

Malgré cette planification, nous avons rencontré quelques difficultés, notamment en matière de communication et d'organisation, en raison des disponibilités variables des membres de l'équipe. Pour pallier cela, nous avons adopté des outils collaboratifs comme Github pour le suivi des tâches et Discord pour les discussions asynchrones. La complexité de certaines requêtes SQL a également posé des défis, nécessitant une décomposition en sous-étapes et une consultation approfondie de la documentation.

De plus, la gestion des contraintes d'intégrité et l'optimisation des performances ont exigé plusieurs ajustements, que nous avons résolus en procédant à des tests progressifs sur des jeux de données simulées

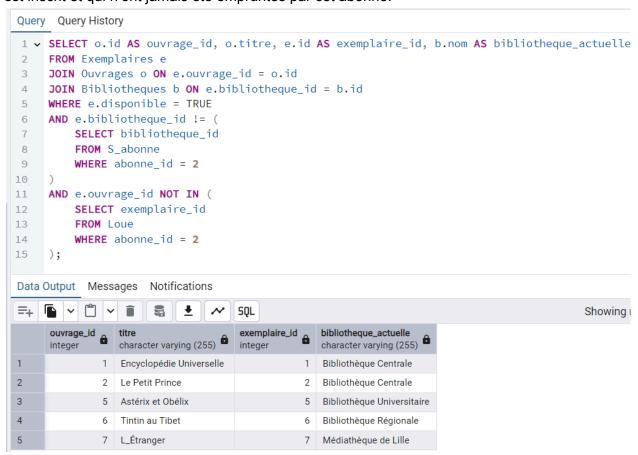
Malgré ces obstacles, plusieurs éléments ont contribué à la réussite du projet. La structuration initiale du travail a facilité la progression et l'intégration des différentes parties du projet. L'utilisation d'outils collaboratifs a permis une gestion plus fluide des tâches et une communication efficace. La reformulation de la globalité du sujet a également joué un rôle clé en facilitant les ajustements et la compréhension des différentes étapes. De plus, une validation rigoureuse et des tests continus ont permis d'identifier rapidement les erreurs et d'optimiser les performances et la structure de la base de données.

Si nous devions recommencer ce projet, nous apporterions certaines améliorations, notamment une meilleure estimation du temps nécessaire pour chaque tâche, en accordant une attention particulière aux phases de test et d'optimisation des performances. Il serait également pertinent de définir des points de contrôle intermédiaires plus précis afin d'éviter l'accumulation de tâches en fin de projet. L'intégration d'analyses de performances et d'indexation dès les premières étapes permettrait également d'améliorer l'efficacité globale du système. Enfin, un suivi plus régulier, avec des réunions hebdomadaires, comme cela a été fait, aiderait à mieux répartir la charge de travail et à anticiper les difficultés. Dans l'ensemble, ce projet a été une expérience enrichissante qui a permis d'acquérir des compétences en gestion de projet et en

optimisation de bases de données, tout en mettant en lumière les axes d'amélioration pour une organisation plus efficace à l'avenir.

## Jeu d'instruction:

 Trouver les ouvrages disponibles dans des bibliothèques autres que celle où l'abonné est inscrit et qui n'ont jamais été empruntés par cet abonné.



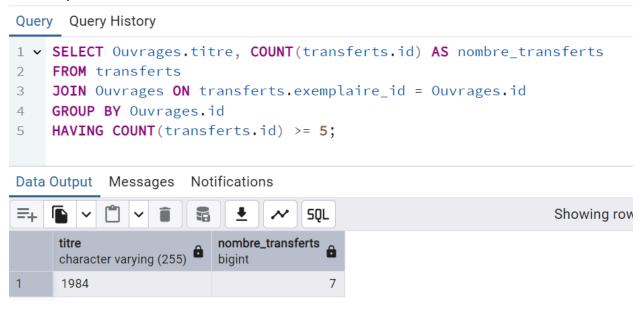
Empêcher qu'un livre emprunté soit prolongé plus de deux fois.

```
livre emprunté soit prolongé plus de deux fois
CREATE OR REPLACE FUNCTION Limiter_prolongations()
RETURNS TRIGGER AS $$
   IF NEW.prolongations >= 2 THEN
     RAISE EXCEPTION 'Le livre ne peut être prolongé plus de 2 fois.';
   RETURN NEW;
$$ LANGUAGE plpgsql;
CREATE TRIGGER avant_prolongation
BEFORE UPDATE ON Loue
FOR EACH ROW
EXECUTE FUNCTION Limiter_prolongations();
Query Query History
1 v UPDATE Loue
      SET prolongations = 1
2
      WHERE abonne_id = 3 AND exemplaire_id = 4;
3
4
      Select * from loue;
5
Data Output
                Messages
                               Notifications
                                                    SQL
                                                                                                Sł
=+
        abonne_id
                         exemplaire_id
                                                                                prolongations
                                            date_emprunt
                                                               date_retour
        [PK] integer
                         [PK] integer
                                            date
                                                                                integer
                                                               date
                                            2025-02-06
                                                               2025-02-20
                                                                                               0
2
                                            2025-02-05
                                                                                               1
                     3
                                                               2025-02-28
```

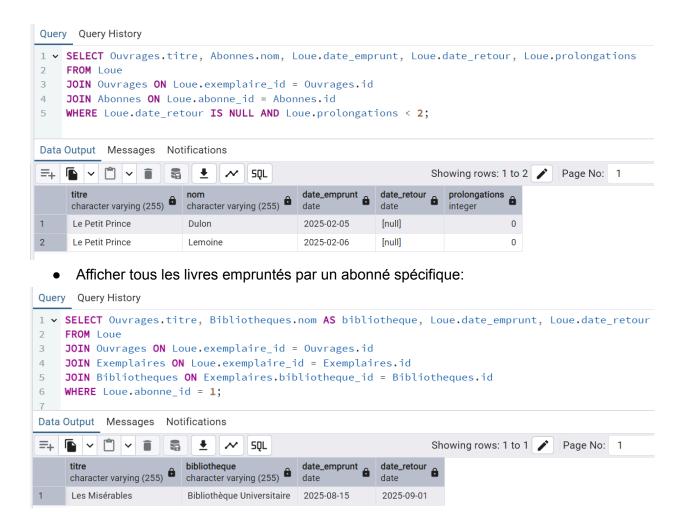
Récupérer la liste des ouvrages les plus empruntés (empruntés plus de 5 fois):

```
Query Query History
1 v CREATE VIEW ouvrages_populaires AS
     SELECT Ouvrages.titre, COUNT(Loue.exemplaire_id) AS nombre_emprunts
2
     FROM Loue
3
4
     JOIN Ouvrages ON Loue.exemplaire_id = Ouvrages.id
5
     GROUP BY Ouvrages.id
     HAVING COUNT(Loue.exemplaire_id) > 5;
6
7
     SELECT * FROM ouvrages_populaires;
Data Output Messages Notifications
                                     SQL.
                                                                     Showing rows:
                         nombre_emprunts
     character varying (255)
                         bigint
     Les Misérables
```

 Afficher la liste des ouvrages ayant été transférés au moins 5 fois entre différentes bibliothèques:



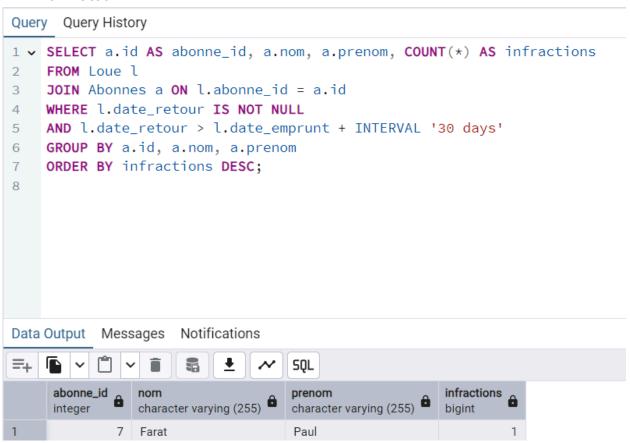
 Afficher la liste des livres actuellement empruntés mais pas encore retournés, à condition que l'abonné ait moins de 2 prolongations:



• Fonction qui calcule le nombre de jours restants avant le retour prévu d'un exemplaire emprunté:

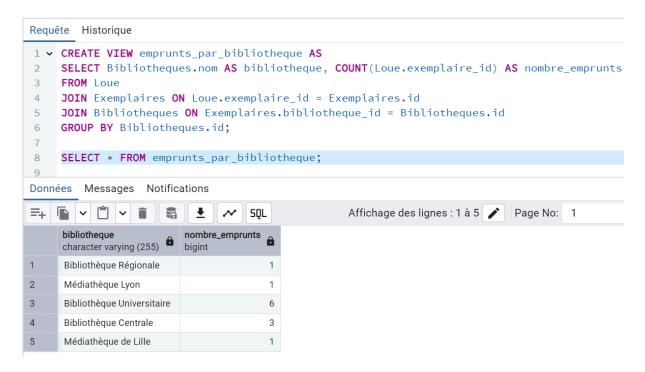


 Quels abonnés ne respectent pas les délais de prêt et quelle est leur fréquence d'infraction:



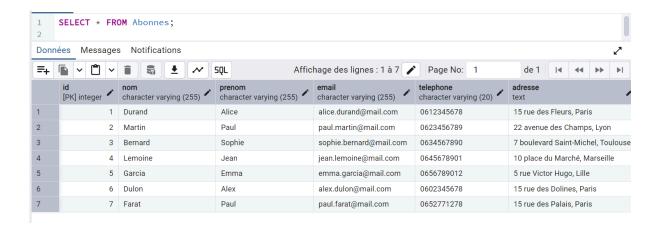
• Création de la vue : emprunts par bibliothèque

Cela retournera une liste des bibliothèques avec le nombre d'emprunts pour chaque bibliothèque.



• Création d'une fonction et d'un déclencheur (trigger) pour mettre à jour le nombre d'emprunts des abonnés

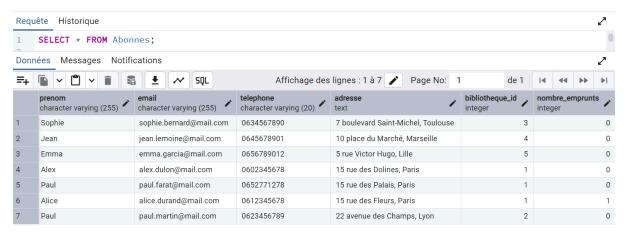
Table Abonnes:

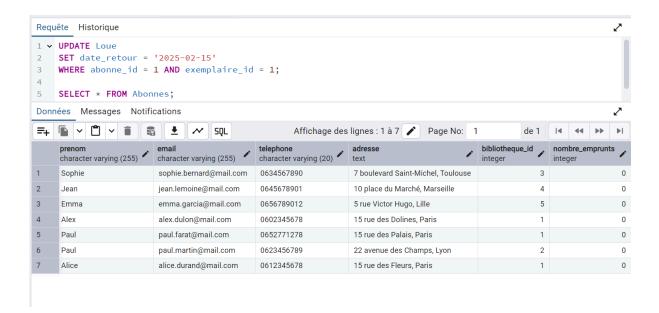


 Cette fonction et le déclencheur (trigger) suivent le nombre d'emprunts effectués par les abonnés. Chaque fois qu'un emprunt est ajouté ou mis à jour dans la table Loue, la fonction est exécutée pour ajuster le champ nombre\_emprunts de la table Abonnes en fonction de l'état de la date de retour (date\_retour).

```
Requête Historique
1 • CREATE OR REPLACE FUNCTION mettre_a_jour_nombre_emprunts()
2
     RETURNS TRIGGER AS $$
3
4
         IF NEW.date_retour IS NULL THEN
 5
             UPDATE Abonnes
6
              SET nombre_emprunts = nombre_emprunts + 1
 7
             WHERE id = NEW.abonne_id;
8
9
         IF OLD.date_retour IS NULL AND NEW.date_retour IS NOT NULL THEN
10 🗸
              UPDATE Abonnes
11
              SET nombre_emprunts = nombre_emprunts - 1
12
13
             WHERE id = NEW.abonne_id;
14
         END IF;
15
         RETURN NEW;
16
     END;
17
18
     $$ LANGUAGE plpgsql;
                   Notifications
Données
        Messages
CREATE FUNCTION
Requête exécutée avec succès en 942 msec.
```

```
1 • CREATE OR REPLACE FUNCTION mettre_a_jour_nombre_emprunts()
     RETURNS TRIGGER AS $$
2
     BEGIN
3
4
         IF NEW.date_retour IS NULL THEN
5
             UPDATE Abonnes
6
             SET nombre_emprunts = nombre_emprunts + 1
             WHERE id = NEW.abonne_id;
7
8
         END IF;
9
10 🗸
         IF OLD.date_retour IS NULL AND NEW.date_retour IS NOT NULL THEN
             UPDATE Abonnes
11
12
             SET nombre_emprunts = nombre_emprunts - 1
13
             WHERE id = NEW.abonne_id;
14
         END IF;
15
         RETURN NEW;
16
17
     END;
18
     $$ LANGUAGE plpgsql;
19
20 CREATE TRIGGER mise_a_jour_emprunts
     AFTER INSERT OR UPDATE ON Loue
21
22
     FOR EACH ROW
     EXECUTE FUNCTION mettre_a_jour_nombre_emprunts();
23
```





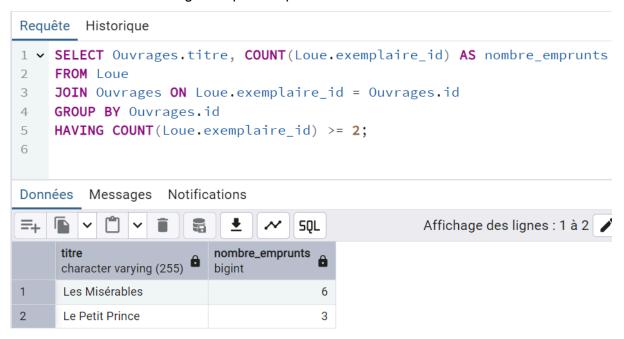
Création de la table Archive Loue et suppression d'anciens emprunts

```
Requête
         Historique
1 ∨ CREATE TABLE Archive_Loue AS
     SELECT * FROM Loue
2
3
     WHERE date_emprunt < CURRENT_DATE - INTERVAL '1 month';</pre>
4
5 V DELETE FROM Loue
     WHERE date_emprunt < CURRENT_DATE - INTERVAL '1 month';</pre>
6
7
8
     SELECT * FROM Archive_Loue;
Données
          Messages
                     Notifications
                                         SQL
                                                           Affichage des ligne
                   exemplaire_id
      abonne_id
                                  date_emprunt
                                                 date_retour
      integer
                   integer
                                  date
                                                 date
1
               1
                               1
                                  2024-01-01
                                                  2024-01-10
```

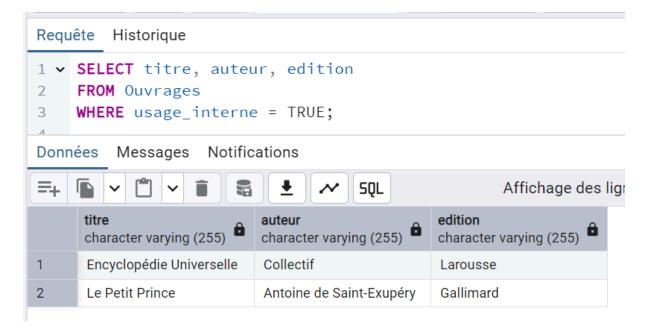
Créer une table Archive\_Loue contenant tous les emprunts plus anciens qu'un mois. Ensuite, les emprunts plus anciens qu'un mois sont supprimés de la table Loue.

Afficher les emprunts archivés et supprimés (table Archive\_Loue et suppression d'emprunts) : Après avoir créé la table Archive\_Loue et supprimé les anciens emprunts dans la table Loue.

Sélection des ouvrages empruntés plus de 2 fois



• Sélection des ouvrages utilisés uniquement en interne



sélectionne les titres, auteurs et éditions des ouvrages dont l'attribut usage\_interne est défini à TRUE (utilisation interne uniquement).

Affiche les ouvrages réservés à l'usage interne.

• Vérification de la disponibilité d'un livre avant un emprunt:

```
Requête Historique
1 v CREATE OR REPLACE FUNCTION check_book_availability()
2 RETURNS TRIGGER AS $$
   BEGIN
3
        IF (SELECT COUNT(*) FROM Reserve WHERE exemplaire_id = NEW.exemplaire_id) > 0 THEN
            RAISE EXCEPTION 'Le livre est déjà réservé.';
6
       END IF;
       RETURN NEW;
7
8 END;
9 $$ LANGUAGE plpgsql;
10
11 • CREATE TRIGGER check_availability
12 BEFORE INSERT ON Loue
13 FOR EACH ROW
14 EXECUTE FUNCTION check_book_availability();
Données Messages Notifications
CREATE TRIGGER
Requête exécutée avec succès en 97 msec.
```

Cette fonction et le déclencheur empêchent l'emprunt d'un livre déjà réservé. Avant d'ajouter un emprunt à la table Loue, le déclencheur vérifie si le livre est déjà réservé en utilisant la table Reserve.

test:

```
INSERT INTO Loue (abonne_id, exemplaire_id, date_emprunt, date_retour)
VALUES (2, 1, '2025-02-02', '2025-02-10');

Données Messages Notifications

ERREUR: Le livre est déjà réservé.
CONTEXT: fonction PL/pgSQL check_book_availability(), ligne 4 à RAISE

ERREUR: Le livre est déjà réservé.
État SQL : P0001
```

1	SELECT * FROM Loue;			
Données Messages Notifications				
=+	~ <u> </u>		<b>✓</b> SQL	Afficha
	abonne_id [PK] integer	exemplaire_id [PK] integer	date_emprunt date	date_retour date
1	5	7	2025-02-06	2025-02-20
2	3	4	2025-02-05	2025-02-28
3	4	4	2025-03-05	2025-03-28
4	6	4	2025-04-05	2025-03-29
5	7	4	2025-06-09	2025-07-28
6	2	4	2025-08-05	2025-08-13
7	1	4	2025-08-15	2025-09-01
8	6	6	2025-02-05	2025-02-28
9	2	3	2025-02-05	2025-02-28
10	7	2	2025-02-05	2025-02-28
11	6	2	2025-02-05	[null]
12	4	2	2025-02-06	[null]

Insertion d'une nouvelle bibliothèque et de ses exemplaires

Cette requête insère une nouvelle bibliothèque dans la base de données et copie tous les ouvrages existants dans la nouvelle bibliothèque en définissant un emplacement par défaut et en les marquant comme disponibles.

Sélection des événements programmés dans une bibliothèque



sélectionne les événements programmés dans une bibliothèque spécifique, en filtrant les événements par bibliothèque id.

Création des rôle et privilège

```
CREATE ROLE bibliothecaire_user LOGIN PASSWORD 'motdepasse';
GRANT SELECT, INSERT, UPDATE, DELETE ON Ouvrages TO bibliothecaire_user;
GRANT SELECT, INSERT, UPDATE ON Loue TO bibliothecaire_user;

CREATE ROLE gestionnaire_user LOGIN PASSWORD 'motdepasse';
GRANT ALL PRIVILEGES ON DATABASE bibliotheque TO gestionnaire_user;

CREATE ROLE admin_user LOGIN PASSWORD 'admin123';
GRANT ALL PRIVILEGES ON DATABASE bibliotheque TO admin_user;

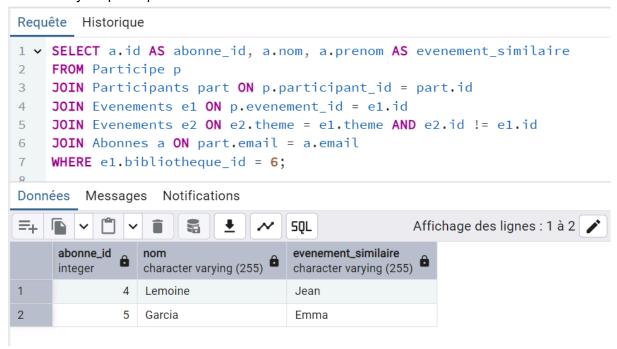
Data Output Messages Notifications

GRANT

Query returned successfully in 53 msec.
```

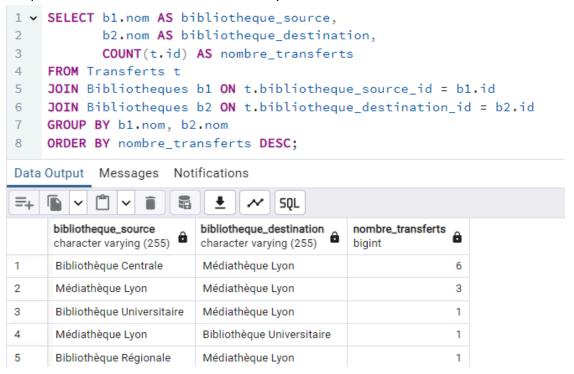
Cette requête permet de créer 3 rôles représentant un abonné, un gestionnaire de bibliothèque et l'admin de la base de données.

• Abonnés ayant participé à des événements similaires



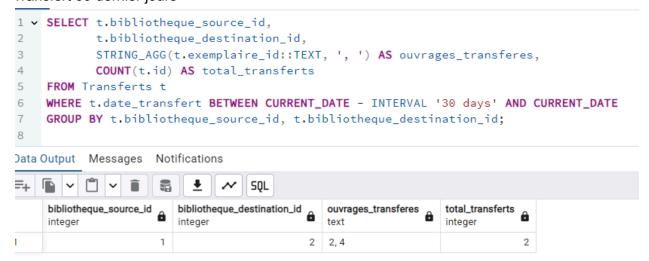
sélectionne les abonnés ayant participé à des événements ayant un thème similaire dans la même bibliothèque.

Fréquence de transfert entre les bibliothèques



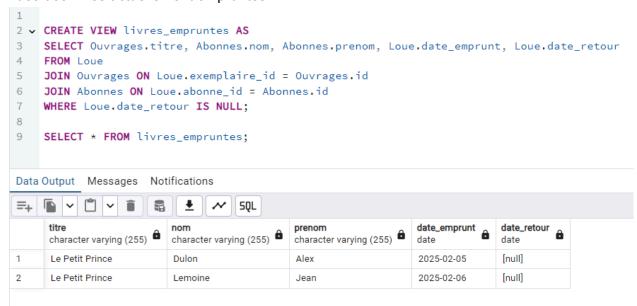
Cette requête nous montre la fréquence d'une bibliothèque source vers une bibliothèque destination, par exemple on voit que la bibliothèque centrale envoie très fréquemment des livres vers la médiathèque de Lyon

Transfert 30 dernier jours



Cette requête permet de récupérer les transfert sur les 30 derniers jours,

• Vues des livres actuellement empruntés



lci 2 livres sont empruntés actuellement, attention ceci est normal un ouvrage peut avoir plusieurs exemplaire dans la bibliothèque.