Rapport Final - Projet BIBLIOTech

Projet de Base de Données Relationnelles - SI3 FISA

Membres du groupe : Luc, Marie, Thomas, Marine

Table des Matières

1. Spécification Complète des Besoins
2. Expression et Explication des Requêtes SQL
3. Tests et Validation
4. Description du Jeu de Données de Test
5. Analyse Critique du Déroulement du Projet

1. Spécification Complète des Besoins

1.1 Contexte et Objectifs

BIBLIOTech est un système de gestion pour un réseau de bibliothèques réparties sur plusieurs régions. Le système doit permettre la gestion centralisée du catalogue, des abonnés, des prêts, des réservations et des transferts d'ouvrages entre établissements.

1.2 Services Rendus par la Base de Données

1.2.1 Gestion du Réseau de Bibliothèques

Le système gère un réseau de bibliothèques organisées par régions géographiques.

Entités et règles métier :

- Une **région** est identifiée par un identifiant unique et possède un nom.
- Une **bibliothèque** appartient à exactement une région et possède un nom et une adresse.
- Les **distances** entre bibliothèques d'une même région sont enregistrées en minutes de transport. Cette information est symétrique : si la distance de A vers B est de 30 minutes, alors la distance de B vers A est également de 30 minutes. Seules les bibliothèques d'une même région peuvent avoir une distance enregistrée entre elles

1.2.2 Gestion du Catalogue et des Ouvrages

Le catalogue recense l'ensemble des ouvrages disponibles dans le réseau.

Entités et règles métier :

- Un **ouvrage** est une œuvre intellectuelle identifiée par un titre, un type de support (Livre, CD, DVD, Revue) et une catégorie (genre littéraire).
- Une **catégorie** regroupe les ouvrages par genre (Roman, Science-Fiction, Histoire, etc.).
- Un **auteur** peut avoir écrit plusieurs ouvrages, et un ouvrage peut avoir plusieurs auteurs (relation N:N).
- Le **catalogue** d'une bibliothèque indique quels ouvrages sont référencés dans cette bibliothèque (un ouvrage peut être présent dans plusieurs bibliothèques).
- Chaque ouvrage possède un **compteur de demandes d'achat** qui s'incrémente lorsqu'un abonné suggère son acquisition.

1.2.3 Gestion du Stock Physique

Les exemplaires sont les copies physiques des ouvrages.

Entités et règles métier :

- Un **exemplaire** est identifié par un code-barre unique. Il correspond à un ouvrage et est localisé dans une bibliothèque précise.
- Chaque exemplaire possède une localisation physique (étage, rayon) et un état (Neuf, Bon, Usé, Endommagé).
- Un exemplaire ne peut appartenir qu'à une seule bibliothèque à un instant donné.

1.2.4 Gestion des Abonnés et Adhésions

Les abonnés sont les usagers du réseau de bibliothèques.

Entités et règles métier :

- Un **abonné** est identifié par un identifiant unique et possède un nom.
- Chaque abonné est associé à un **type d'abonnement** (Étudiant, Professeur, Lambda) qui détermine :
    - Le **quota maximum** d'emprunts simultanés autorisés
    - La **durée maximale** d'un prêt en jours
- Un abonné peut être **bloqué** suite à un retard de restitution. Le blocage a une date de fin calculée en fonction du nombre de jours de retard.
- Un abonné bloqué ne peut pas effectuer de nouveaux emprunts jusqu'à la fin de son blocage.

1.2.5 Gestion des Prêts

Les prêts permettent aux abonnés d'emprunter des exemplaires.

Entités et règles métier :

- Un **emprunt** lie un abonné à un exemplaire pour une période donnée.
- La **date de retour prévue** est calculée automatiquement : date d'emprunt + durée maximale selon le type d'abonnement.
- Un emprunt peut être **prolongé** une seule fois (booléen est_prolonge).
- Un abonné ne peut pas emprunter s'il est bloqué ou s'il a atteint son quota.
- Un même abonné peut emprunter le même exemplaire plusieurs fois (à des dates différentes), d'où la clé primaire composite (id_abonne, code_barre, date_emprunt).

1.2.6 Gestion des Réservations

Les réservations permettent de réserver un ouvrage indisponible.

Entités et règles métier :

- Une **réservation** porte sur un ouvrage (pas un exemplaire spécifique) et indique la bibliothèque de retrait souhaitée.
- Les réservations sont gérées en **file d'attente FIFO** (premier arrivé, premier servi) basée sur l'horodatage de la demande.
- Lorsqu'un exemplaire est retourné :
    - S'il y a des réservations en attente pour cet ouvrage, le premier de la file est notifié
    - Si la bibliothèque de retour diffère de celle souhaitée, un transfert est déclenché
- Un abonné ne peut avoir qu'une seule réservation active par ouvrage.

1.2.7 Gestion des Suggestions d'Achat

Les abonnés peuvent suggérer l'acquisition de nouveaux ouvrages.

Entités et règles métier :

- Une **suggestion** lie un abonné à un ouvrage avec une date.
- Chaque suggestion incrémente le compteur de demandes de l'ouvrage.
- Lorsque le compteur atteint un seuil défini, une alerte est générée pour les gestionnaires.

1.2.8 Gestion des Transferts

Les transferts permettent de déplacer des exemplaires entre bibliothèques.

Entités et règles métier :

- Un **transfert** concerne un exemplaire et indique la bibliothèque source, la bibliothèque destination et le statut (En transit, Reçu, Annulé).
- Le temps de transport estimé est calculé à partir de la table des distances (ESTT_DISTANT).
- Un transfert peut être déclenché automatiquement suite à une réservation ou manuellement par le personnel.

1.2.9 Gestion des Événements

Les bibliothèques organisent des événements culturels.

Entités et règles métier :

- Un **événement** a une date, une capacité maximale et est organisé par une bibliothèque.
- Les **abonnés** peuvent s'inscrire nominativement (table PARTICIPE).
- Les **non-abonnés** peuvent également participer mais de manière anonyme (compteur nb_inscrits).
- Le nombre de places restantes = capacité - abonnés inscrits - non-abonnés inscrits.

2. Expression et Explication des Requêtes SQL

(on a eu un peu de mal pour mettre en forme cette partie dans le document)

Cette section présente l'ensemble des requêtes SQL développées pour le projet, organisées par membre et par module fonctionnel.

2.1 Luc : Module "Catalogue et Statistiques"

Notions SQL avancées choisies : CTE (Common Table Expressions) et Fonctions de fenêtrage (ROW_NUMBER, PARTITION BY)

Requête L1 : Recherche d'ouvrages par titre
Rédigé par : Luc

```
SELECT o.id_ouvrage, o.titre, o.type_support, c.libelle_genre AS categorie, a.nom AS auteur

FROM OUVRAGE o

JOIN CATEGORIE c ON o.id_cat = c.id_cat

JOIN ECRIT e ON o.id_ouvrage = e.id_ouvrage

JOIN AUTEUR a ON e.id_auteur = a.id_auteur

WHERE o.titre LIKE '%mot_recherche%'

ORDER BY o.titre;
```

Explication : Cette requête permet une recherche textuelle flexible sur le titre des ouvrages. Elle utilise une série de JOIN pour agréger les informations depuis les tables OUVRAGE, CATEGORIE, ECRIT et AUTEUR, ce qui est nécessaire en raison de la normalisation de la base. L'opérateur LIKE avec les jokers % permet de trouver des correspondances partielles, rendant la recherche plus conviviale pour l'utilisateur. Le ORDER BY final assure que les résultats sont présentés de manière alphabétique.

Requête L2 : Recherche d'ouvrages par auteur

Rédigé par : Luc

```
SELECT o.id_ouvrage, o.titre, o.type_support, c.libelle_genre AS categorie, a.nom AS auteur

FROM OUVRAGE o
```

```
JOIN CATEGORIE c ON o.id_cat = c.id_cat

JOIN ECRIT e ON o.id_ouvrage = e.id_ouvrage

JOIN AUTEUR a ON e.id_auteur = a.id_auteur

WHERE a.nom LIKE '%nom_auteur%'

ORDER BY o.titre;
```

Explication : similaire à la recherche par titre, cette requête filtre les résultats en fonction du nom de l'auteur. La clause WHERE s'applique ici sur a.nom, démontrant la capacité du modèle à être interrogé sur différentes facettes des données. La structure des jointures reste identique, soulignant la réutilisabilité de ce pattern de lecture.

Requête L3 : Recherche d'ouvrages par catégorie
Rédigé par : Luc

```
SELECT o.id_ouvrage, o.titre, o.type_support, c.libelle_genre AS categorie

FROM OUVRAGE o

JOIN CATEGORIE c ON o.id_cat = c.id_cat

WHERE c.libelle_genre = 'Roman'

ORDER BY o.titre;
```

Explication : Cette requête est optimisée pour la recherche par catégorie. Contrairement aux précédentes, elle ne nécessite pas de joindre les tables ECRIT et AUTEUR.. Elle montre l'importance de n'inclure que les jointures strictement nécessaires à la résolution d'une question.

Requête L4 : Affichage de la fiche ouvrage avec ses exemplaires

Rédigé par : Luc

```
SELECT o.titre, o.type_support, c.libelle_genre AS categorie,

    ex.code_barre, ex.etat, ex.etage, ex.rayon,

    b.nom AS bibliotheque

FROM OUVRAGE o

JOIN CATEGORIE c ON o.id_cat = c.id_cat

JOIN EXEMPLAIRE ex ON o.id_ouvrage = ex.id_ouvrage

JOIN BIBLIOTHEQUE b ON ex.id_biblio = b.id_biblio

WHERE o.id_ouvrage = 1;
```

Explication : Cette requête est cruciale pour l'utilisateur final car elle fournit une vue complète d'un ouvrage et de toutes ses copies physiques disponibles dans le réseau. Elle agrège les informations de l'œuvre (titre, catégorie), de l'exemplaire (état, localisation) et de la bibliothèque (nom). Une même requête peut retourner plusieurs lignes si un ouvrage possède plusieurs exemplaires, ce qui est le comportement attendu.

Requête L5 : Enregistrer une suggestion d'achat

Rédigé par : Luc

```
-- Début de la transaction

BEGIN;

INSERT INTO SUGGERE (id_abonne, id_ouvrage, date_suggestion)

VALUES (1, 5, CURRENT_DATE);

UPDATE OUVRAGE

SET compteur_demande_achat = compteur_demande_achat + 1

WHERE id_ouvrage = 5;
```

-- Fin de la transaction

COMMIT;

Explication : Cette opération illustre une transaction métier. Elle est composée de deux opérations atomiques : l'enregistrement de la suggestion dans la table SUGGERE et l'incrémentation du compteur dans la table OUVRAGE. L'utilisation d'une transaction explicite (BEGIN; ... COMMIT;) est fondamentale ici pour garantir l'intégrité des données : si l'une des deux requêtes échoue, l'autre est automatiquement annulée (ROLLBACK), évitant un état incohérent de la base.

Requête L6 : Alerter quand le seuil de suggestions est atteint

Rédigé par : Luc

SELECT id_ouvrage, titre, compteur_demande_achat

FROM OUVRAGE

WHERE compteur_demande_achat >= 5

ORDER BY compteur_demande_achat DESC;

Explication : Il s'agit d'une requête de reporting simple mais essentielle pour la gestion du catalogue. Elle permet aux bibliothécaires d'identifier rapidement les ouvrages les plus demandés par les abonnés et de prendre des décisions d'achat éclairées. Le ORDER BY permet de prioriser les ouvrages ayant le plus de demandes.

Requête L7 : Rapport de popularité par région (TÂCHE DIFFICILE)

Rédigé par : Luc

Notions avancées utilisées : CTE (Common Table Expressions) et Fonctions de fenêtrage (Window Functions)

```
WITH EmpruntsParRegion AS (
    SELECT
        r.id_region,
        r.nom_region,
        o.id_ouvrage,
        o.titre,
        COUNT(*) AS nb_emprunts
    FROM EMPRUNTE em
    JOIN EXEMPLAIRE ex ON em.code_barre = ex.code_barre
    JOIN OUVRAGE o ON ex.id_ouvrage = o.id_ouvrage
    JOIN BIBLIOTHEQUE b ON ex.id_biblio = b.id_biblio
    JOIN REGION r ON b.id_region = r.id_region
    WHERE em.date_emprunt >= CURRENT_DATE - INTERVAL '6 months'
    GROUP BY r.id_region, r.nom_region, o.id_ouvrage, o.titre
),
ClassementParRegion AS (
    SELECT
        id_region,
        nom_region,
```

```
        id_ouvrage,

        titre,

        nb_emprunts,

        ROW_NUMBER() OVER(PARTITION BY id_region ORDER BY nb_emprunts DESC) AS
rang

    FROM EmpruntsParRegion

)

SELECT

    nom_region,

    titre,

    nb_emprunts

FROM ClassementParRegion

WHERE rang <= 3;
```

Explication : Cette requête complexe génère un rapport du TOP 3 des ouvrages les plus populaires dans chaque région sur les 6 derniers mois.

1. La première CTE EmpruntsParRegion agrège le nombre d'emprunts par ouvrage et par région. C'est une étape de préparation des données.
2. La seconde CTE ClassementParRegion utilise la fonction de fenêtrage ROW_NUMBER() pour assigner un rang à chaque ouvrage au sein de sa région (PARTITION BY id_region), basé sur le nombre d'emprunts (ORDER BY nb_emprunts DESC).
3. La requête finale sélectionne simplement les ouvrages dont le rang est inférieur ou égal à 3, fournissant ainsi le TOP 3 demandé. L'utilisation des CTE rend la requête lisible et maintenable malgré sa complexité :)

2.2 Marie : Module "Réservations et Événements"

Notions SQL avancées choisies : Requêtes imbriquées, Fonctions de fenêtrage, CTE (Common Table Expressions)

Requête M1 : Créer une réservation

Rédigé par : Marie

```
INSERT INTO RESERVE (id_abonne, id_ouvrage, date_demande, id_biblio_retrait, statut)

VALUES (2, 3, NOW(), 2, 'En attente');
```

Explication : Requête d'insertion standard qui crée une nouvelle réservation. Elle enregistre qui réserve (id_abonne), quoi (id_ouvrage), quand (date_demande avec NOW()), et où le retrait est souhaité (id_biblio_retrait). Le statut est initialisé à 'En attente', ce qui est le point de départ du cycle de vie de la réservation.

Requête M2 : Consulter les réservations d'un abonné

Rédigé par : Marie

```
SELECT o.titre, r.date_demande, b.nom AS bibliotheque_retrait, r.statut

FROM RESERVE r

JOIN OUVRAGE o ON r.id_ouvrage = o.id_ouvrage

JOIN BIBLIOTHEQUE b ON r.id_biblio_retrait = b.id_biblio

WHERE r.id_abonne = 1;
```

Explication : Permet à un abonné de consulter ses réservations en cours. La requête joint les tables OUVRAGE et BIBLIOTHEQUE pour afficher des informations lisibles par l'homme (titre de l'ouvrage, nom de la bibliothèque) plutôt que de simples identifiants.

Requête M3 : Afficher la file d'attente pour un ouvrage

Rédigé par : Marie

```
SELECT
    a.nom AS abonne,
    r.date_demande,
    ROW_NUMBER() OVER (ORDER BY r.date_demande ASC) as position_file_attente
FROM RESERVE r
JOIN ABONNE a ON r.id_abonne = a.id_abonne
WHERE r.id_ouvrage = 3 AND r.statut = 'En attente'
ORDER BY r.date_demande ASC;
```

Explication : Cette requête implémente la règle métier de la file d'attente FIFO. Elle utilise la fonction de fenêtrage ROW_NUMBER() pour calculer dynamiquement la position de chaque abonné dans la file, en se basant sur la date de la demande (ORDER BY r.date_demande ASC). C'est une méthode efficace et standard en SQL pour gérer les classements.

Requête M4 : Notifier le prochain abonné dans la file

Rédigé par : Marie

```
UPDATE RESERVE
SET statut = 'Disponible'
WHERE (id_abonne, id_ouvrage) = (
    SELECT id_abonne, id_ouvrage
    FROM RESERVE
    WHERE id_ouvrage = 3 AND statut = 'En attente'
    ORDER BY date_demande ASC
```

```
    LIMIT 1
);
```

Explication : Cette requête met à jour le statut de la réservation pour le premier abonné de la file d'attente. Elle utilise une sous-requête qui identifie la réservation la plus ancienne (ORDER BY date_demande ASC et LIMIT 1) pour un ouvrage donné. C'est une manière atomique et sûre de faire progresser la file d'attente.

Requête M5 : Logique de décision post-retour (TÂCHE DIFFICILE)

Rédigé par : Marie

Notions avancées utilisées : CTE (Common Table Expressions) et CASE WHEN

```
WITH ProchaineReservation AS (
    SELECT id_biblio_retrait
    FROM RESERVE
    WHERE id_ouvrage = 1 AND statut = 'En attente'
    ORDER BY date_demande ASC
    LIMIT 1
)
SELECT
    CASE
        WHEN (SELECT id_biblio_retrait FROM ProchaineReservation) IS NULL
            THEN 'Remettre en rayon'
        WHEN (SELECT id_biblio_retrait FROM ProchaineReservation) = 1 -- id_biblio_retour
            THEN 'Mettre de côté pour réservation'
```

```
    ELSE 'Transférer vers la bibliothèque ' || (SELECT id_biblio_retrait FROM
ProchaineReservation)

  END AS action_requise;
```

Explication : Cette requête complexe détermine l'action à effectuer lorsqu'un exemplaire est retourné.

1. La CTE ProchaineReservation identifie la prochaine réservation en attente pour l'ouvrage concerné.
2. La requête principale utilise une instruction CASE WHEN pour implémenter la logique de décision :
    - Si aucune réservation n'existe (IS NULL), l'action est de remettre l'exemplaire en rayon.
    - Si la bibliothèque de retrait de la réservation est la même que la bibliothèque de retour, l'action est de mettre l'exemplaire de côté.
    - Sinon, l'action est de déclencher un transfert vers la bibliothèque de retrait souhaitée. Cette approche centralise une règle métier complexe en une seule requête déclarative.

Requête M6 : Créer un événement

Rédigé par : Marie

```
INSERT INTO EVENEMENT (nom_event, date_event, capacite_max, id_biblio)

VALUES ('Rencontre avec l''auteur', '2026-03-15 18:00:00', 50, 1);
```

Explication : Requête d'insertion simple pour créer un nouvel événement. Notez l'utilisation de l'apostrophe doublée (l''auteur) pour échapper le caractère spécial en SQL standard.

Requête M7 : Inscrire un abonné à un événement

Rédigé par : Marie

```
INSERT INTO PARTICIPE (id_event, id_abonne)

VALUES (1, 1);
```

Explication : Crée une inscription nominative pour un abonné à un événement via la table de liaison PARTICIPE. C'est une opération standard pour une relation N:N.

Requête M8 : Inscrire un non-abonné (anonyme)

Rédigé par : Marie

UPDATE EVENEMENT

SET nb_inscrits_non_abonnes = nb_inscrits_non_abonnes + 1

WHERE id_event = 1;

Explication : Gère les inscriptions anonymes en incrémentant simplement un compteur dans la table EVENEMENT. Cette approche évite de devoir créer des entrées inutiles pour des participants non identifiés.

2.3 Thomas : Module "Abonnés, Prêts et Sanctions"

Notions SQL avancées choisies : Fonctions PL/pgSQL, Sous-requêtes corrélées

Requête T1 : Lister les abonnés

Rédigé par : Thomas

SELECT a.id_abonne, a.nom, t.libelle_type, a.est_bloque, a.fin_blocage

FROM ABONNE a

JOIN TYPE_ABONNEMENT t ON a.id_type = t.id_type

ORDER BY a.nom;

Explication : Fournit une vue d'ensemble de tous les abonnés, en joignant la table TYPE_ABONNEMENT pour afficher le libellé du type d'abonnement plutôt que son simple ID. C'est une requête de base pour l'administration des utilisateurs.

Requête T2 : Afficher la fiche détaillée d'un abonné

Rédigé par : Thomas

SELECT a.nom, t.libelle_type, t.quota_max, t.duree_max_pret

FROM ABONNE a

```
JOIN TYPE_ABONNEMENT t ON a.id_type = t.id_type

WHERE a.id_abonne = 1;
```

Explication : Affiche les droits et informations spécifiques d'un abonné. Cette requête est essentielle pour le personnel de la bibliothèque afin de vérifier le statut et les permissions d'un usager.

Requête T3 : Lister les types d'abonnement

Rédigé par : Thomas

```
SELECT libelle_type, quota_max, duree_max_pret

FROM TYPE_ABONNEMENT;
```

Explication : Requête simple pour afficher les différentes formules d'abonnement proposées par le réseau de bibliothèques. Utile pour la configuration du système et pour informer les nouveaux usagers.

Requête T4 : Vérification des droits avant un emprunt

Rédigé par : Thomas

```
SELECT

  CASE

    WHEN a.est_bloque = TRUE AND a.fin_blocage > CURRENT_DATE THEN 'KO - Abonné bloqué'

    WHEN (

      SELECT COUNT(*)

      FROM EMPRUNTE

      WHERE id_abonne = 1

    ) >= (SELECT quota_max FROM TYPE_ABONNEMENT WHERE id_type = a.id_type)
```

```
        THEN 'KO - Quota atteint'

    ELSE 'OK - Emprunt autorisé'

  END AS statut_emprunt

FROM ABONNE a

WHERE a.id_abonne = 1;
```

Explication : Cette requête est une implémentation directe des règles métier liées à l'emprunt. Elle utilise une instruction CASE WHEN pour effectuer une série de vérifications :

1. L'abonné est-il actuellement bloqué ?
2. L'abonné a-t-il déjà atteint son quota d'emprunts simultanés ? (ceci est vérifié via une sous-requête corrélée qui compte les emprunts actifs de l'abonné). Le résultat est un message clair indiquant si l'emprunt peut avoir lieu ou non.

Requête T5 : Enregistrer un nouvel emprunt

Rédigé par : Thomas

```
INSERT INTO EMPRUNTE (id_abonne, code_barre, date_emprunt, date_retour_prevue)

VALUES (1, 'XYZ789', CURRENT_DATE, CURRENT_DATE + (SELECT duree_max_pret
FROM TYPE_ABONNEMENT WHERE id_type = 1));
```

Explication : Enregistre un nouvel emprunt. La date_retour_prevue est calculée dynamiquement en ajoutant la durée maximale du prêt (récupérée via une sous-requête sur TYPE_ABONNEMENT) à la date actuelle. Cela garantit que la date de retour est toujours conforme aux règles de l'abonnement.

Requête T6 : Enregistrer un retour

Rédigé par : Thomas

```
DELETE FROM EMPRUNTE

WHERE id_abonne = 1 AND code_barre = 'XYZ789';
```

Explication : La restitution d'un exemplaire est matérialisée par la suppression de l'enregistrement correspondant dans la table EMPRUNTE. C'est une approche simple et efficace pour gérer les prêts actifs.

Requête T7 : Consulter l'historique des prêts d'un abonné

Rédigé par : Thomas

```
-- Note : Pour un véritable historique, il faudrait une table `HISTORIQUE_EMPRUNTS`.

-- En l'état, on ne peut voir que les prêts en cours.

SELECT e.date_emprunt, e.date_retour_prevue, o.titre

FROM EMPRUNTE e

JOIN EXEMPLAIRE ex ON e.code_barre = ex.code_barre

JOIN OUVRAGE o ON ex.id_ouvrage = o.id_ouvrage

WHERE e.id_abonne = 1;
```

Explication : Cette requête permet de voir les emprunts actuellement actifs pour un abonné. J'ai ajouté un commentaire pour souligner une limite de notre modèle actuel : comme les retours sont gérés par une suppression, nous ne conservons pas d'historique des prêts passés. Ce qui serait intéressant: Pour une future version, la création d'une table HISTORIQUE_EMPRUNTS serait nécessaire.

Requête T8 : Lister tous les prêts en retard

Rédigé par : Thomas

```
SELECT a.nom, o.titre, e.date_retour_prevue, (CURRENT_DATE - e.date_retour_prevue) AS
jours_de_retard

FROM EMPRUNTE e

JOIN ABONNE a ON e.id_abonne = a.id_abonne

JOIN EXEMPLAIRE ex ON e.code_barre = ex.code_barre
```

```
JOIN OUVRAGE o ON ex.id_ouvrage = o.id_ouvrage

WHERE e.date_retour_prevue < CURRENT_DATE;
```

Explication : Requête de reporting cruciale pour le personnel. Elle identifie tous les emprunts dont la date de retour est dépassée et calcule le nombre de jours de retard. C'est le point de départ pour l'application des sanctionss.

Requête T9 : Automatisation des sanctions (TÂCHE DIFFICILE)

Rédigé par : Thomas

Notions avancées utilisées : Fonction PL/pgSQL

```
CREATE OR REPLACE FUNCTION retourner_exemplaire(

    p_id_abonne INT,

    p_code_barre VARCHAR,

    p_date_emprunt DATE

)

RETURNS VOID AS $$

DECLARE

    v_date_retour_prevue DATE;

    v_jours_retard INT;

BEGIN

    -- Récupérer la date de retour prévue

    SELECT date_retour_prevue INTO v_date_retour_prevue

    FROM EMPRUNTE

    WHERE id_abonne = p_id_abonne

      AND code_barre = p_code_barre
```

```
      AND date_emprunt = p_date_emprunt;

    -- Calculer le retard (en jours)

    v_jours_retard := CURRENT_DATE - v_date_retour_prevue;

    -- Si retard, appliquer le blocage

    IF v_jours_retard > 0 THEN

        UPDATE ABONNE

        SET est_bloque = TRUE,

            fin_blocage = CURRENT_DATE + (v_jours_retard || ' days')::INTERVAL

        WHERE id_abonne = p_id_abonne;

    END IF;

    -- Supprimer l'emprunt (le livre est rendu)

    DELETE FROM EMPRUNTE

    WHERE id_abonne = p_id_abonne

      AND code_barre = p_code_barre

      AND date_emprunt = p_date_emprunt;

END;

$$ LANGUAGE plpgsql;
```

Explication : Cette fonction PL/pgSQL encapsule toute la logique métier du retour d'un exemplaire. Plutôt que de laisser l'application exécuter plusieurs requêtes séparées, la base de données s'en charge de manière atomique et sécurisée. (Informations tirées de la docu de PostGreSQL)

1. **Déclaration de variables :** v_date_retour_prevue et v_jours_retard sont déclarées pour stocker des valeurs intermédiaires.
2. **Logique procédurale :** La fonction exécute une séquence d'actions : un SELECT pour obtenir une valeur, un calcul, une condition IF pour appliquer ou non une sanction, et enfin un DELETE pour finaliser le retour.
3. **Sécurité et intégrité :** En centralisant cette logique dans la base, on s'assure qu'elle est toujours exécutée de la même manière, quel que soit le client qui s'y connecte. C'est une pratique robuste pour les règles métier critiques.

2.4 Marine : Module "Réseau et Logistique"

Notions SQL avancées choisies : Jointure réflexive, Sous-requêtes dans le SELECT

Requête MA1 : Lister les bibliothèques et leur région

Rédigé par : Marine

```
SELECT b.nom AS bibliotheque, b.adresse, r.nom_region

FROM BIBLIOTHEQUE b

JOIN REGION r ON b.id_region = r.id_region;
```

Explication : Requête de base pour visualiser le maillage territorial du réseau, en associant chaque bibliothèque à sa région de tutelle.

Requête MA2 : Lister les régions

Rédigé par : Marine

```
SELECT nom_region FROM REGION;
```

Explication : Requête très simple pour obtenir la liste de toutes les régions couvertes par le réseau BIBLIOTech.

Requête MA3 : Lister les événements à venir avec places restantes

Rédigé par : Marine

```
SELECT
```

```
    nom_event,

    date_event,

    b.nom AS lieu,

    (capacite_max - (SELECT COUNT(*) FROM PARTICIPE p WHERE p.id_event = e.id_event)
- nb_inscrits_non_abonnes) AS places_restantes

FROM EVENEMENT e

JOIN BIBLIOTHEQUE b ON e.id_biblio = b.id_biblio

WHERE date_event > NOW()

ORDER BY date_event;
```

Explication : Cette requête calcule dynamiquement le nombre de places restantes pour chaque événement. Elle utilise une sous-requête dans la clause SELECT pour compter le nombre d'abonnés inscrits, et y soustrait le nombre de non-abonnés. C'est une information cruciale pour les usagers souhaitant s'inscrire.

Requête MA4 : Lister les participants à un événement

Rédigé par : Marine

```
SELECT a.nom

FROM PARTICIPE p

JOIN ABONNE a ON p.id_abonne = a.id_abonne

WHERE p.id_event = 1;

-- Pour les non-abonnés, on ne peut que donner le nombre

SELECT nb_inscrits_non_abonnes FROM EVENEMENT WHERE id_event = 1;
```

Explication : Pour des raisons de confidentialité et de modélisation, les participants sont gérés de deux manières. Cette première requête liste les noms des abonnés inscrits. Une seconde requête est nécessaire pour connaître le nombre de participants anonymes (non-abonnés).

Requête MA5 : Créer un transfert d'exemplaire

Rédigé par : Marine

```
INSERT INTO TRANSFERT (code_barre, id_biblio_source, id_biblio_destination, date_creation, statut)

VALUES ('JKL345', 1, 2, CURRENT_DATE, 'En transit');
```

Explication : Enregistre le début d'un transfert logistique d'un exemplaire entre deux bibliothèques. Le statut est initialisé à 'En transit'.

Requête MA6 : Suivi des transferts avec temps de transport (TÂCHE DIFFICILE)

Rédigé par : Marine

Notions avancées utilisées : Jointure réflexive

```
SELECT

    t.code_barre,

    o.titre,

    b_source.nom AS source,

    b_dest.nom AS destination,

    t.statut,

    dist.temps_transport AS temps_estime_min

FROM TRANSFERT t

JOIN EXEMPLAIRE ex ON t.code_barre = ex.code_barre

JOIN OUVRAGE o ON ex.id_ouvrage = o.id_ouvrage

JOIN BIBLIOTHEQUE b_source ON t.id_biblio_source = b_source.id_biblio

JOIN BIBLIOTHEQUE b_dest ON t.id_biblio_destination = b_dest.id_biblio
```

```
LEFT JOIN EST_DISTANT dist ON

   (dist.id_biblio_A = t.id_biblio_source AND dist.id_biblio_B = t.id_biblio_destination)

   OR (dist.id_biblio_A = t.id_biblio_destination AND dist.id_biblio_B = t.id_biblio_source);
```

Explication : Cette requête complexe permet de suivre les transferts en cours tout en affichant le temps de transport estimé. La difficulté réside dans la jointure avec la table EST_DISTANT. Comme la distance est symétrique (A vers B = B vers A), il faut utiliser une jointure LEFT JOIN avec une condition OR complexe pour trouver la correspondance, quelle que soit la direction enregistrée dans la table. Les alias b_source et b_dest sont utilisés pour joindre deux fois la table BIBLIOTHEQUE sous des rôles différents.

Requête MA7 : Lister les transferts partant d'une bibliothèque

Rédigé par : Marine

```
SELECT t.code_barre, o.titre, b_dest.nom AS destination, t.statut

FROM TRANSFERT t

JOIN EXEMPLAIRE ex ON t.code_barre = ex.code_barre

JOIN OUVRAGE o ON ex.id_ouvrage = o.id_ouvrage

JOIN BIBLIOTHEQUE b_dest ON t.id_biblio_destination = b_dest.id_biblio

WHERE t.id_biblio_source = 1;
```

Explication : Requête de reporting logistique qui filtre les transferts en fonction de leur bibliothèque d'origine. Utile pour le personnel d'une bibliothèque pour savoir quels exemplaires ont été expédiés.

Requête MA8 : Lister les transferts arrivant à une bibliothèque

Rédigé par : Marine

```
SELECT t.code_barre, o.titre, b_source.nom AS source, t.statut

FROM TRANSFERT t
```

JOIN EXEMPLAIRE ex ON t.code_barre = ex.code_barre

JOIN OUVRAGE o ON ex.id_ouvrage = o.id_ouvrage

JOIN BIBLIOTHEQUE b_source ON t.id_biblio_source = b_source.id_biblio

WHERE t.id_biblio_destination = 2;

Explication : Similaire à la précédente, mais filtre sur la bibliothèque de destination. Permet au personnel de savoir quels exemplaires sont attendus en réception.

3. Tests et Validation

3.1 Procédure de Test

Pour garantir la reproductibilité et la validité de nos tests, nous avons suivi cette procédure en utilisant le client de PostgreSQL:

1. **Connexion à PostgreSQL :**

   psql -U postgres

2. **Création d'un environnement de test propre :** Nous supprimons la base de données de test si elle existe pour garantir un départ à zéro, puis nous la recréons. Cela évite que les résultats des tests précédents n'interfèrent.

   DROP DATABASE IF EXISTS bibliotech_test;

   CREATE DATABASE bibliotech_test;

   \c bibliotech_test

3. **Désactivation du pager :** Pour une lecture fluide des résultats dans le terminal, nous désactivons le pager de psql.

   \pset pager off

4. **Exécution des scripts :** Nous exécutons les trois scripts SQL dans un ordre précis :

    - Le DDL pour créer la structure.
    - Le jeu de données pour peupler la base.
    - et enfin Le fichier de requêtes pour tester les fonctionnalités.

3.2 Résultats d'Exécution et Analyse des Échecs

L'exécution de la procédure ci-dessus a permis de valider l'ensemble des 32 requêtes. Toutes les requêtes SELECT, UPDATE et DELETE se sont comportées comme attendu, et la fonction PL/pgSQL a été créée avec succès.

Le seul "échec" observé est une erreur duplicate key lors de l'exécution de la requête M7 (Inscrire un abonné à un événement).

Analyse de l'erreur :

ERROR: duplicate key value violates unique constraint "participe_pkey"

DETAIL: Key (id_event, id_abonne)=(1, 1) already exists.

Cet échec est en réalité un **succès de validation**. Il prouve que notre contrainte de clé primaire sur la table PARTICIPE (qui empêche un même abonné de s'inscrire plusieurs fois au même événement) fonctionne parfaitement. Le jeu de données insère déjà cette participation, donc la tentative de la réinsérer via le script de requêtes est correctement rejetée par le SGBD !

Conclusion des tests : Le système est robuste et les contraintes d'intégrité sont bien implémentées. Aucun test fonctionnel n'a échoué.

3.3 Validation Détaillée par Module

- **Module Catalogue (Luc) :** Les recherches multicritères (L1-L3) sont fonctionnelles. La fiche ouvrage (L4) agrège correctement les données. La transaction de suggestion (L5) est atomique. Le rapport de popularité (L7) avec CTE et fonctions de fenêtrage classe correctement les ouvrages par région.
- **Module Réservations (Marie) :** La file d'attente FIFO (M3) est respectée. La logique de décision post-retour (M5) fonctionne comme spécifié. La gestion des inscriptions aux événements (M6-M8) est correcte.
- **Module Prêts (Thomas) :** Les vérifications de droits (T4) et la gestion des quotas sont fonctionnelles. La fonction PL/pgSQL de sanction (T9) est créée et s'exécute, appliquant la logique de blocage en cas de retard.
- **Module Logistique (Marine) :** La jointure réflexive pour le suivi des transferts (MA6) calcule bien le temps de transport. Les requêtes de reporting (MA3, MA7, MA8) filtrent correctement les données.

4. Description du Jeu de Données de Test

Le jeu de données a été conçu pour être minimaliste mais exhaustif, afin de couvrir les cas d'usage nominaux et les cas limites de chaque fonctionnalité.

Table	**Nombre d'enregistrements**	**Justification de la complétude**
REGION	3	Permet de tester les requêtes multi-régions.
BIBLIOTHEQUE	4	Assez pour tester les transferts et les recherches localisées.
CATEGORIE	5	Couvre plusieurs genres littéraires.
AUTEUR	5	Permet des ouvrages avec un ou plusieurs auteurs.
OUVRAGE	6	Inclut des ouvrages avec et sans exemplaires.
EXEMPLAIRE	8	Certains ouvrages ont plusieurs exemplaires, d'autres aucun.
TYPE_ABONNEMENT	3	Différents quotas et durées de prêt pour tester les règles.
ABONNE	4	Inclut un abonné déjà bloqué pour tester les sanctions.
EMPRUNTE	3	Contient des prêts en cours pour tester les quotas et retours.
RESERVE	3	Crée une file d'attente sur un ouvrage pour tester le FIFO.
SUGGERE	2	Permet de tester l'incrémentation du compteur.

Table	**Nombre d'enregistrements**	**Justification de la complétude**
EVENEMENT	3	Événements avec et sans inscrits, passés et futurs.
PARTICIPE	2	Inscriptions nominatives.
TRANSFERT	2	Transferts en cours pour tester le suivi.
EST_DISTANT	2	Distances entre bibliothèques pour le calcul du temps.

Ce jeu de données permet de valider :

- Les jointures sur toutes les tables.
- Les calculs (places restantes, jours de retard).
- Les cas limites (abonné bloqué, quota atteint, file d'attente).
- Les logiques complexes (sanctions, décision post-retour, popularité).

5. Analyse Critique du Déroulement du Projet

5.1 Découpage et Répartition du Travail

L'approche choisie, consistant à découper le projet en modules fonctionnels et à attribuer la pleine propriété de chaque module à un membre, s'est avérée très efficace. Cette méthode a favorisé l'autonomie et la responsabilisation. La répartition équilibrée des tâches, avec une tâche difficile clairement identifiée pour chacun, a permis à tous les membres de monter en compétence sur des notions SQL avancées.

5.2 Difficultés Rencontrées

Le principal obstacle a été la **migration de la syntaxe MySQL vers PostgreSQL** en milieu de projet. Nous avions initialement commencé à écrire des requêtes en nous basant sur MySQL, et la transition a nécessité une réécriture significative, notamment pour :

- Les fonctions de date (CURDATE() vs CURRENT_DATE).
- La syntaxe des intervalles de temps.
- La syntaxe des procédures stockées, qui est très différente avec PL/pgSQL.

Cette difficulté imprévue a consommé du temps mais a été une expérience d'apprentissage précieuse sur l'importance de la portabilité du code SQL.

5.3 Aspects Bien Réussis

- **Solidité du modèle de données :** Le MCD initial était bien conçu, et le MLD qui en a découlé a nécessité très peu d'ajustements. La normalisation en 3FN a prouvé sa robustesse.
- **Implémentation des logiques complexes :** Chaque membre a réussi à implémenter sa tâche difficile, démontrant une bonne compréhension des notions SQL avancées (CTE, fonctions de fenêtrage, PL/pgSQL, jointures réflexives).
- **Collaboration et communication :** L'entraide a été constante, notamment lors de la phase de débogage des requêtes et de la migration vers PostgreSQL.

5.4 Améliorations Possibles

Si nous devions refaire le projet, nous apporterions deux améliorations majeures à notre organisation :

1. **Valider l'environnement technique cible dès le premier jour :** Le choix du SGBD (PostgreSQL) aurait dû être acté et testé par toute l'équipe dès le début. Cela nous aurait évité la phase de migration.
2. **Écrire les tests en parallèle :** Plutôt que de tester toutes les requêtes à la fin, nous aurions pu valider chaque requête dès son écriture. Cela aurait permis de détecter les problèmes de syntaxe plus tôt et de manière plus isolée.

En conclusion, malgré les défis techniques, l'organisation modulaire et la bonne communication ont été les clés de la réussite de ce projet.