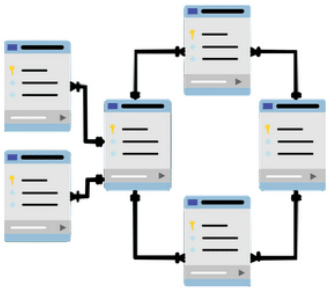
Rapport de projet

INTRODUCTION AUX BASES DE DONNEES

INFO3Bb



Gestion des Données des Péages d'Autoroute : Modélisation, Implémentation et Analyse

Par Brice LAWSON

SOMMAIRE

1. Introduction

Dans le cadre de ce projet, nous avons pour objectif de concevoir et de mettre en place un système de gestion des péages autoroutiers, en utilisant une base de données relationnelle, un langage de programmation Java et des requêtes SQL. Le système doit permettre de gérer diverses fonctionnalités liées aux utilisateurs, aux trajets, aux abonnements et aux péages.

Contexte

Les péages autoroutiers sont une source de revenus importante pour les autorités routières et, pour garantir une gestion efficace, il est crucial d'avoir un système capable de suivre et de gérer en temps réel les passages, les paiements, ainsi que les abonnements des automobilistes. Ce projet vise à développer un système complet pour gérer ces opérations, tout en permettant l'analyse des comportements des automobilistes. En particulier, il nous est demandé de répondre à plusieurs questions fonctionnelles, telles que :

- Identifier les péages les plus empruntés.
- Détail des trajets d'un utilisateur abonné spécifique.
- Détecter les automobilistes non abonnés ayant des trajets répétés, afin de proposer des campagnes de promotion pour les abonnements.
- Identifier les portions d'autoroute les plus utilisées.
- Lister les automobilistes dont l'abonnement est expiré mais qui conservent un badge actif.

Objectifs

Le projet se divise en plusieurs étapes, chacune ayant des objectifs précis :

1. **Modélisation des données : Créer une base de données relationnelle qui permettra de stocker toutes les informations nécessaires (utilisateurs, péages, trajets, abonnements, etc.).**
2. **Traduction de la modélisation en SQL : Rédiger les scripts SQL nécessaires à la création de la base de données, ainsi que les requêtes permettant d'extraire des informations pertinentes.**
3. **Développement d'une application Java : Créer une interface Java permettant d'interagir avec la base de données, d'exécuter des requêtes et de présenter les résultats sous une forme lisible pour l'utilisateur.**
4. **Analyse des données : Utiliser les requêtes SQL pour répondre aux questions posées par le projet et générer des rapports sur l'utilisation des péages, les trajets récurrents, les portions d'autoroute les plus fréquentées, et les abonnements expirés.**

Méthodologie

Le projet s'est appuyé sur une méthodologie itérative, où chaque composant (base de données, application Java, et requêtes SQL) a été développé, testé et amélioré au fur et à mesure. L'utilisation de technologies éprouvées telles que PostgreSQL pour la gestion de la base de données et Java pour l'interface permet de garantir la performance et la robustesse du système.

2. Analyse fonctionnelle

Le sujet nous demande de développer un système de gestion des péages autoroutiers permettant de répondre à diverses requêtes sur les automobilistes, les trajets effectués, ainsi que les abonnements aux péages. L'objectif est de pouvoir analyser les comportements des automobilistes, d'identifier ceux qui utilisent fréquemment certains trajets sans être abonnés, de repérer les portions d'autoroute les plus empruntées, et de détecter les automobilistes dont l'abonnement est expiré mais qui conservent un badge actif.

Les principales fonctionnalités demandées sont les suivantes :

1. **Identification des péages les plus empruntés** : Nous devons fournir une liste des péages les plus fréquentés, afin de mieux comprendre où se concentrent les flux de trafic. Cela pourrait être utile pour la gestion de la circulation et la planification des infrastructures.
2. **Liste des trajets d'un utilisateur abonné** : Pour un automobiliste donné, nous devons être capables de lister tous les trajets effectués sur une période donnée, avec des détails sur les péages d'entrée et de sortie, ainsi que le montant facturé.
3. **Détection des automobilistes non abonnés ayant des trajets répétés** : Une analyse des trajets répétés effectués par des automobilistes non abonnés doit être réalisée pour cibler des campagnes de promotion. Ces automobilistes pourraient être incités à souscrire à un abonnement, ce qui permettrait de simplifier leur expérience et de générer des revenus récurrents pour le gestionnaire de péage.
4. **Identification des portions d'autoroute les plus empruntées** : Nous devons déterminer quelles portions d'autoroute (composées de péages d'entrée et de sortie) sont les plus fréquentées. Cela peut fournir des informations précieuses pour l'optimisation de la gestion du réseau autoroutier.
5. **Automobilistes avec badge actif mais abonnement expiré** : Certains automobilistes pourraient continuer à utiliser leur badge après l'expiration de leur abonnement. Il est important de détecter ces cas afin d'envoyer des rappels de renouvellement d'abonnement ou de suspendre l'accès si nécessaire.

Hypothèses et compléments apportés au sujet

Dans le cadre de cette analyse fonctionnelle, plusieurs hypothèses ont été formulées pour compléter et clarifier le sujet :

1. **Fréquence des trajets répétés** : Nous considérons qu'un "trajet répété" est un trajet effectué à plusieurs reprises entre les mêmes péages d'entrée et de sortie, sans distinction du nombre de jours entre ces trajets. Par exemple, un automobiliste effectuant plusieurs fois le même trajet dans une même semaine sera pris en compte.
2. **Abonnements et badges** : Nous avons supposé que les abonnements ont une durée fixe, avec une date de début et une date de fin. De plus, chaque automobiliste peut posséder un badge actif, qui lui permet de passer les péages sans payer directement. L'existence de cet abonnement est vérifiée par le statut "abonné" dans la base de données.
3. **Données de trafic** : Pour les péages les plus fréquentés et les portions d'autoroute les plus empruntées, il est supposé que la fréquence de passage à un péage est

enregistrée dans la base de données, soit par chaque ticket émis, soit par un compteur de passages associé à chaque péage.

4. **Utilisation de la base de données** : Nous avons considéré que toutes les données nécessaires (utilisateurs, péages, tickets, abonnements, badges) sont présentes et correctement renseignées dans la base de données, ce qui permet l'exécution des requêtes SQL nécessaires.
5. **Traitement des dates** : Nous avons également pris l'hypothèse que la gestion des dates (dates d'entrée, de sortie, d'abonnement) est correcte et permet une analyse cohérente, notamment pour identifier les abonnements expirés ou les trajets effectués après l'expiration de l'abonnement.

Objectifs fonctionnels du système

Le système doit permettre de :

- **Analyser le trafic autoroutier** : En identifiant les péages les plus fréquentés et les portions d'autoroute les plus utilisées, ce système aide à mieux comprendre les comportements de circulation sur le réseau.
- **Cibler les campagnes de promotion** : En détectant les automobilistes non abonnés qui empruntent fréquemment les mêmes trajets, il devient possible de les cibler avec des offres spécifiques pour les encourager à souscrire à un abonnement.
- **Gérer les abonnements et les badges** : Le système doit suivre l'état des abonnements et des badges, afin de prévenir les automobilistes lorsque leur abonnement expire et de détecter les badges encore actifs après expiration.

Cette analyse fonctionnelle permet de poser les bases de la modélisation des données et du développement du système. Elle met en évidence les questions auxquelles le système doit répondre, les hypothèses sur lesquelles il repose, ainsi que les besoins spécifiques en matière de gestion des péages et des abonnements. Le développement ultérieur du modèle Entité-Association et la traduction de celui-ci en SQL seront guidés par cette analyse.

3. Modélisation du sujet avec le modèle Entité-Association

Approche méthodologique

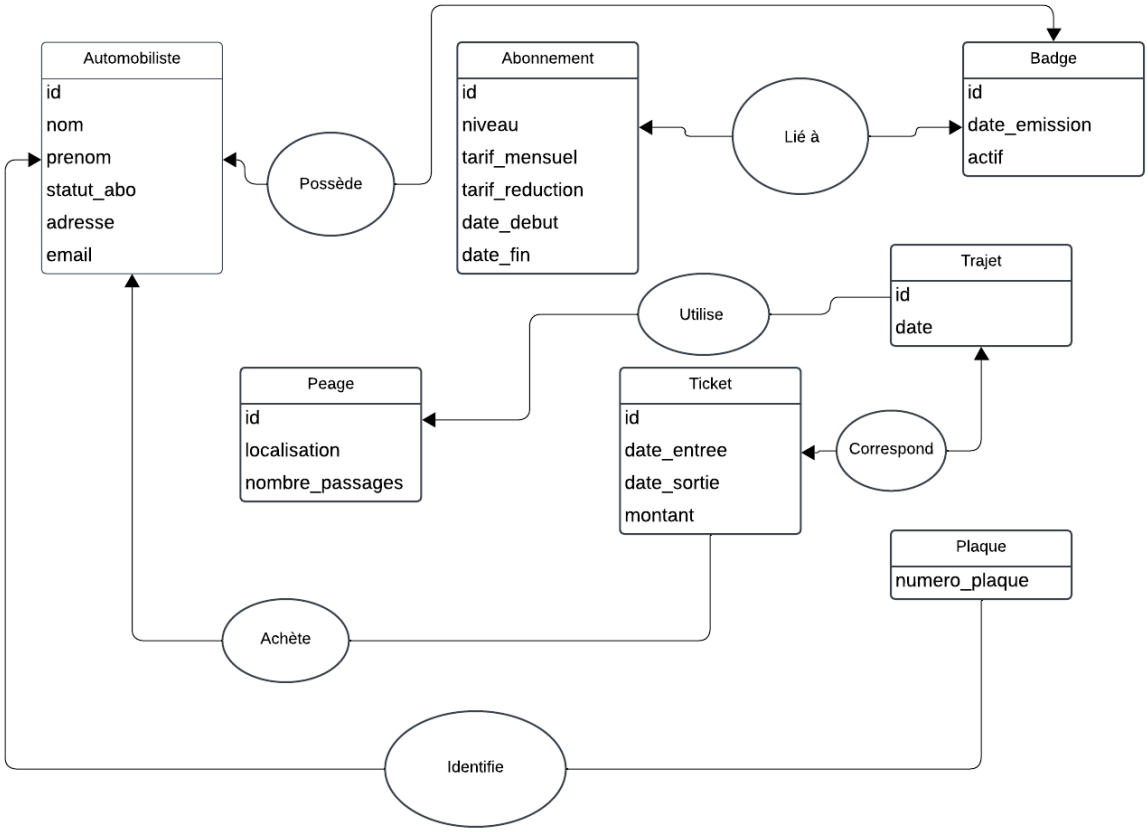
Le modèle Entité-Association (EA) permet de représenter de manière visuelle et structurée les données, leurs interrelations, et les contraintes associées. Le système de gestion des péages repose sur plusieurs entités principales telles que les **Automobilistes**, les **Péages**, les **Tickets**, les **Abonnements**, et les **Badges**. Ces entités interagissent entre elles pour répondre aux questions fonctionnelles décrites dans l'analyse précédente.

4 : Justification de la modélisation

La modélisation des données repose sur une approche centrée sur les besoins fonctionnels du système de gestion des péages. Voici les choix effectués ainsi que leurs justifications :

Diagramme Entité-Association

Le diagramme EA correspondant illustre clairement les entités et leurs relations. Il met en évidence la modélisation logique utilisée pour structurer les données et répondre aux besoins fonctionnels du projet.



Entités et leurs Relations

- **Automobiliste** :
  - **Relation avec Abonnement** : Un automobiliste peut avoir **un seul abonnement** (relation 1,1).
  - **Relation avec Badge** : Un automobiliste peut posséder **un seul badge** (relation 1,1).
  - **Relation avec Ticket** : Un automobiliste peut acheter **plusieurs tickets** (relation 1,N).
  - **Relation avec Plaque** : Un automobiliste peut avoir **plusieurs plaques** (relation 1,N).
- **Abonnement** :
  - Lié à **Automobiliste** par id_automobiliste (relation 1,1).
- **Badge** :
  - Lié à **Automobiliste** par id_automobiliste (relation 1,1).
- **Peage** :

    - **Relation avec Ticket** : Un péage peut être lié à **plusieurs tickets** via peage_entree et peage_sortie (relation 1,N).
    - **Relation avec Trajet** : Un péage peut être lié à **plusieurs trajets** via id_peage_entree et id_peage_sortie (relation 1,N).
- **Ticket** :
    - **Relation avec Automobiliste** : Un ticket est acheté par **un seul automobiliste,** mais un automobiliste peut avoir **plusieurs tickets** (relation 1,N).
    - **Relation avec Peage** : Un ticket est lié à **deux péages** (entrée et sortie), chaque péage pouvant être lié à **plusieurs tickets** (relation 1,N).
    - **Relation avec Trajet** : Un ticket correspond à **un seul trajet** (relation 1,1).
- **Trajet** :
    - **Relation avec Ticket** : Un trajet est associé à **un seul ticket** (relation 1,1).
    - **Relation avec Peage** : Un trajet implique **deux péages** (entrée et sortie), chaque péage pouvant être lié à **plusieurs trajets** (relation 1,N).
- **Plaque** :
    - **Relation avec Automobiliste** : Une plaque appartient à **un seul automobiliste,** mais un automobiliste peut avoir **plusieurs plaques** (relation 1,N).

1. **Organisation par schéma** : L'utilisation du schéma Gestion_peages permet de centraliser toutes les tables liées au projet, assurant une meilleure lisibilité et gestion.
2. **Relations entre entités** :
    - Chaque **Automobiliste** peut avoir un **Abonnement,** un **Badge,** et plusieurs **Tickets,** permettant de suivre leurs interactions avec le système.
    - Les **Tickets** sont associés aux **Peages** d'entrée et de sortie, reflétant le parcours réel des trajets.
    - La table **Plaque** est liée directement à l'Automobiliste pour gérer les informations des véhicules de manière indépendante.
3. **Choix des cardinalités** :
    - Les relations de type (1,1) ou (1,N) ont été choisies en fonction de la logique métier (par exemple, un automobiliste peut générer plusieurs tickets, mais chaque ticket est lié à un seul trajet).
4. **Flexibilité et évolutivité** :
    - Les structures permettent d'ajouter de nouveaux types d'abonnements, de péages ou d'automobilistes sans impacter les relations existantes.

Cette modélisation équilibre simplicité et robustesse, tout en facilitant l'extraction des données nécessaires pour répondre aux questions métier.

5. Requêtes SQL principales

Cette section présente les requêtes SQL utilisées pour répondre aux différentes questions du sujet.

5.1 Création du schéma et des tables

Pour organiser les données, un schéma nommé gestion_peages est créé, suivi des tables nécessaires :

```sql
-- Création du schéma pour organiser les tables du projet
CREATE SCHEMA gestion_peages;
```

Tables principales

1. **Table des automobilistes :**

```sql
-- Table des informations des automobilistes

CREATE TABLE gestion_peages.Automobiliste (
    id SERIAL PRIMARY KEY,        -- Identifiant unique pour chaque automobiliste
    nom TEXT,                     -- Nom de l'automobiliste
    prenom TEXT,                  -- Prénom de l'automobiliste
    statut_abo VARCHAR(10),       -- Indique si l'automobiliste est "abonne" ou "non-abonne"
    adresse TEXT,                 -- Adresse postale de l'automobiliste
    email VARCHAR(100)            -- Adresse e-mail de l'automobiliste
);
```

- Contient les informations personnelles des automobilistes, avec un champ statut_abo pour indiquer s'ils sont abonnés ou non.

2. **Table des abonnements :**

```sql
-- Table des informations des abonnements

CREATE TABLE gestion_peages.Abonnement (
    id SERIAL PRIMARY KEY,              -- Identifiant unique pour chaque abonnement
    niveau VARCHAR(50),                 -- Niveau d'abonnement (par exemple, "Standard", "Premium")
    tarif_mensuel NUMERIC(10, 2),       -- Tarif mensuel pour le prêt du badge
    tarif_reduction NUMERIC(10, 2),     -- Pourcentage de réduction appliqué sur les trajets
    date_debut DATE,                    -- Date de début de l'abonnement
    date_fin DATE,                      -- Date de fin de l'abonnement
    id_automobiliste INT REFERENCES gestion_peages.Automobiliste(id) -- Lien avec l'automobiliste concerné
);
```

- Stocke les détails des abonnements, comme les dates et les tarifs.

3. **Table des badges :**

```sql
-- Table des badges d'identification des abonnés

CREATE TABLE gestion_peages.Badge (
    id SERIAL PRIMARY KEY,              -- Identifiant unique pour chaque badge
    id_automobiliste INT REFERENCES gestion_peages.Automobiliste(id), -- Automobiliste possédant le badge
    date_emission DATE,                 -- Date d'émission du badge
    actif BOOLEAN DEFAULT TRUE          -- Indique si le badge est actif (par défaut, TRUE)
);
```

- Permet de gérer les badges électroniques des automobilistes.

4. **Table des péages :**

```sql
-- Table des informations des péages

CREATE TABLE gestion_peages.Peage (
    id SERIAL PRIMARY KEY,              -- Identifiant unique pour chaque péage
    localisation TEXT,                  -- Localisation du péage
    nombre_passages INT DEFAULT 0       -- Nombre total de passages enregistrés (par défaut, 0)
);

```

- Gère les informations des péages et le nombre de passages.

5. **Table des tickets :**

```sql
-- Table des tickets émis pour les trajets des automobilistes

CREATE TABLE gestion_peages.Ticket (
    id SERIAL PRIMARY KEY,              -- Identifiant unique pour chaque ticket
    date_entree TIMESTAMP,              -- Date et heure d'entrée sur l'autoroute
    peage_entree INT REFERENCES Peage(id), -- Péage d'entrée (lien avec la table Peage)
    date_sortie TIMESTAMP,              -- Date et heure de sortie de l'autoroute
    peage_sortie INT REFERENCES Peage(id), -- Péage de sortie (lien avec la table Peage)
    montant NUMERIC(10, 2),             -- Montant facturé pour le trajet
    id_automobiliste INT REFERENCES gestion_peages.Automobiliste(id) -- Automobiliste concerné
);

```

- Enregistre les trajets des automobilistes avec leurs montants.

6. **Table des trajets :**

```sql
-- Table des trajets réalisés

CREATE TABLE gestion_peages.Trajet (
    id SERIAL PRIMARY KEY,              -- Identifiant unique pour chaque trajet
    id_ticket INT REFERENCES Ticket(id),-- Lien vers le ticket associé au trajet
    id_peage_entree INT REFERENCES Peage(id), -- Péage d'entrée
    id_peage_sortie INT REFERENCES Peage(id), -- Péage de sortie
    date DATE,                          -- Date du trajet
    id_automobiliste INT REFERENCES gestion_peages.Automobiliste(id) -- Automobiliste ayant effectué le trajet
);

```

- Utilisée pour regrouper les informations de trajets effectués.

7. **Table des plaques :**

```sql
-- Table des plaques des véhicules pour des raisons de sécurité

CREATE TABLE gestion_peages.Plaque (
    numero_plaque VARCHAR(15) PRIMARY KEY, -- Numéro de plaque d'immatriculation unique
    id_automobiliste INT REFERENCES gestion_peages.Automobiliste(id) -- Automobiliste possédant le véhicule
);

```

- Lie les automobilistes à leurs plaques d'immatriculation.

5.2 Justifications des choix

- **Schéma dédié** : Le schéma gestion_peages garantit une organisation claire et évite les conflits avec d'autres tables.
- **Relations entre les entités** : Les clés étrangères (e.g., id_automobiliste) permettent de lier les données efficacement tout en respectant les contraintes d'intégrité référentielle.
- **Sécurité des données** : Les types de données et les contraintes (e.g., PRIMARY KEY, FOREIGN KEY) assurent une validation stricte et une cohérence des informations.

5.3 Avantages de la modélisation

- **Extensibilité** : De nouvelles tables ou colonnes peuvent être ajoutées sans affecter les relations existantes.
- **Efficacité** : Les relations bien définies permettent des requêtes SQL optimisées.

Le script complet est fourni en annexe pour référence et mise en œuvre.

6. Explication du code Java

Le code Java permet d'interagir avec la base de données PostgreSQL afin de répondre aux questions du sujet. Voici une synthèse des éléments techniques principaux :

6.1 Connexion à la base de données

La connexion est établie via la classe DriverManager. Les informations (URL, utilisateur, mot de passe) sont centralisées pour simplifier la configuration :

```
1 usage
private static final String URL = "jdbc:postgresql://kafka.iem/rl646187";
1 usage
private static final String USER = "rl646187";
1 usage
private static final String PASSWORD = "rl646187";

public static void main(String[] args) {
    try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD)) {
        System.out.println("Connexion réussie à la base de données.");
```

6.2 Menu principal

Le menu interactif guide l'utilisateur vers différentes fonctionnalités, avec un traitement des choix via un switch :

```
while (continuer) {
    System.out.println("\nOptions disponibles :");
    System.out.println("1. Afficher les péages les plus empruntés");
    System.out.println("2. Liste des trajets d'un utilisateur abonné");
    System.out.println("3. Automobilistes non abonnés avec trajets répétés");
    System.out.println("4. Portions d'autoroute les plus empruntées");
    System.out.println("5. Automobilistes avec badge et abonnement expiré");
    System.out.println("6. Quitter");
    System.out.print("Votre choix : ");
```

6.3 Méthodes principales

a) Afficher les péages les plus empruntés

Exécute une requête SQL pour récupérer les péages les plus fréquentés :

```
1 usage
private static void afficherPeagesLesPlusEmpruntes(Connection connection) {
    String query = """
        SELECT localisation, nombre_passages
        FROM Gestion_peages.Peage
        ORDER BY nombre_passages DESC
        LIMIT 10;
    """;
    try (Statement statement = connection.createStatement();
         ResultSet resultSet = statement.executeQuery(query)) {

        System.out.println("\nPéages les plus empruntés :");
        while (resultSet.next()) {
            System.out.println("Localisation : " + resultSet.getString( columnLabel: "localisation") +
                    ", Nombre de passages : " + resultSet.getInt( columnLabel: "nombre_passages"));
        }
    } catch (SQLException e) {
        System.err.println("Erreur lors de la requête : " + e.getMessage());
    }
}
```

b) Liste des trajets d'un utilisateur abonné

Utilisation d'une requête préparée pour récupérer les trajets sécurisés par un identifiant utilisateur :

```
1 usage
private static void afficherTrajetsUtilisateurAbonne(Connection connection, int idUtilisateur) {
    String query = """
    SELECT t.date_entree, p1.localisation AS peage_entree, p2.localisation AS peage_sortie, t.montant
    FROM Gestion_peages.Ticket t
    JOIN Gestion_peages.Peage p1 ON t.peage_entree = p1.id
    JOIN Gestion_peages.Peage p2 ON t.peage_sortie = p2.id
    WHERE t.id_automobiliste = ?;
""";
    try (PreparedStatement preparedStatement = connection.prepareStatement(query)) {
        preparedStatement.setInt( parameterIndex: 1, idUtilisateur);

        try (ResultSet resultSet = preparedStatement.executeQuery()) {
            System.out.println("\nTrajets de l'utilisateur abonné :");
            if (!resultSet.isBeforeFirst()) { // Si aucun résultat
                System.out.println("Aucun trajet trouvé pour cet utilisateur.");
            } else {
                while (resultSet.next()) {
                    System.out.println("Date d'entrée : " + resultSet.getTimestamp( columnLabel: "date_entree") +
                            ", Péage d'entrée : " + resultSet.getString( columnLabel: "peage_entree") +
                            ", Péage de sortie : " + resultSet.getString( columnLabel: "peage_sortie") +
                            ", Montant : " + resultSet.getDouble( columnLabel: "montant"));
```

```
                }
            }
        } catch (SQLException e) {
            System.err.println("Erreur lors de la requête : " + e.getMessage());
        }
    }
```

c) Automobilistes non abonnés avec trajets répétés

```
1 usage
private static void detecterAutomobilistesNonAbonnes(Connection connection) {
    String query = """
        SELECT a.nom, a.prenom, COUNT(*) AS nb_trajets
        FROM Gestion_peages.Ticket t
        JOIN Gestion_peages.Automobiliste a ON t.id_automobiliste = a.id
        WHERE a.statut_abo = 'non-abonne'
        GROUP BY a.id, a.nom, a.prenom
        HAVING COUNT(*) > 1;
    """;
    try (Statement statement = connection.createStatement();
         ResultSet resultSet = statement.executeQuery(query)) {

        System.out.println("\nAutomobilistes non abonnés avec trajets répétés :");
        while (resultSet.next()) {
            System.out.println("Nom : " + resultSet.getString( columnLabel: "nom") +
                    ", Prénom : " + resultSet.getString( columnLabel: "prenom") +
                    ", Nombre de trajets : " + resultSet.getInt( columnLabel: "nb_trajets"));
        }
    } catch (SQLException e) {
        System.err.println("Erreur lors de la requête : " + e.getMessage());
    }
}
```

6.5 Points forts

- **Modularité** : Méthodes distinctes pour chaque fonctionnalité.
- **Sécurité** : Utilisation de requêtes préparées pour éviter les injections SQL.
- **Simplicité** : Menu clair et interface utilisateur conviviale.

Cette approche garantit une exécution fiable et sécurisée. Le code complet est fourni en annexe pour plus de détails.

7. Requêtes SQL principales

Les requêtes SQL suivantes répondent aux différentes problématiques définies dans le sujet. Voici une description des éléments principaux :

7.1 Péages les plus empruntés

Cette requête utilise une agrégation pour compter le nombre de passages par péage :

```sql
--Quels sont les peages les plus empruntes ?
SELECT p.localisation, COUNT(t.id) AS nombre_passages
FROM Gestion_peages.Peage p
JOIN Gestion_peages.Ticket t ON p.id = t.peage_entree OR p.id = t.peage_sortie
GROUP BY p.localisation
ORDER BY nombre_passages DESC
LIMIT 10;
```

Explication technique :

- Utilisation de ORDER BY pour trier les résultats par ordre décroissant du nombre de passages.
- Limitation du nombre de résultats à 10 grâce à LIMIT.

7.2 Liste des trajets d'un utilisateur abonné

Cette requête retourne les trajets effectués par un utilisateur abonné :

```sql
--Pour un utilisateur abonne donne en parametre, lui retourner la liste de ses trajets
SELECT t.date_entree, p1.localisation AS peage_entree, p2.localisation AS peage_sortie, t.montant
FROM Gestion_peages.Ticket t
JOIN Gestion_peages.Peage p1 ON t.peage_entree = p1.id
JOIN Gestion_peages.Peage p2 ON t.peage_sortie = p2.id
WHERE t.id_automobiliste = ?;
```

Explication technique :

- Les jointures JOIN relient les tables Ticket et Peage pour obtenir les informations sur les péages d'entrée et de sortie.
- Le paramètre ? est utilisé pour sécuriser la requête avec des valeurs dynamiques fournies par l'utilisateur.

7.3 Automobilistes non abonnés avec trajets répétés

Pour identifier les trajets fréquents des non-abonnés :

```sql
--Quels sont les automobilistes non abonnes qui font souvent le meme trajet
SELECT a.nom, a.prenom, p1.localisation AS peage_entree, p2.localisation AS peage_sortie, COUNT(*) AS nb_trajets
FROM Gestion_peages.Ticket t
JOIN Gestion_peages.Automobiliste a ON t.id_automobiliste = a.id
JOIN Gestion_peages.Peage p1 ON t.peage_entree = p1.id
JOIN Gestion_peages.Peage p2 ON t.peage_sortie = p2.id
WHERE a.statut_abo = 'non-abonne'
GROUP BY a.id, a.nom, a.prenom, p1.localisation, p2.localisation
HAVING COUNT(*) > 1
ORDER BY nb_trajets DESC;
```

Explication technique :

- GROUP BY permet d'agréger les données par automobiliste.
- La clause HAVING filtre les résultats pour ne conserver que ceux ayant effectué plus d'un trajet.

7.4 Portions d'autoroute les plus empruntées

Cette requête identifie les portions les plus fréquentées :

```sql
--Quelles sont les portions d'autoroute les plus empruntees
SELECT p1.localisation AS peage_entree, p2.localisation AS peage_sortie, COUNT(*) AS nb_passages
FROM Gestion_peages.Ticket t
JOIN Gestion_peages.Peage p1 ON t.peage_entree = p1.id
JOIN Gestion_peages.Peage p2 ON t.peage_sortie = p2.id
GROUP BY p1.localisation, p2.localisation
ORDER BY nb_passages DESC
LIMIT 10;
```

Explication technique :

- Les jointures permettent de récupérer les localisations des péages d'entrée et de sortie.
- COUNT(*) compte le nombre de trajets pour chaque portion.
- Tri décroissant des portions selon leur fréquence avec ORDER BY.

7.5 Automobilistes avec abonnement expiré et badge actif

Cette requête identifie les utilisateurs ayant un abonnement expiré mais un badge actif :

```sql
--Quels sont les automobilistes dont l'abonnement est termine mais qui ont encore en leur possession un badge ?
SELECT a.nom, a.prenom, b.id AS badge_id
FROM Gestion_peages.Automobiliste a
JOIN Gestion_peages.Abonnement ab ON a.id = ab.id_automobiliste
JOIN Gestion_peages.Badge b ON a.id = b.id_automobiliste
WHERE ab.date_fin < CURRENT_DATE AND b.actif = TRUE;

```

Explication technique :

- CURRENT_DATE est utilisé pour vérifier si la date de fin de l'abonnement est passée.
- Filtrage des badges actifs avec la condition b.actif = TRUE.

7.6 Points forts des requêtes

- **Optimisation** : Les requêtes utilisent des index via les clés primaires et étrangères.
- **Lisibilité** : Utilisation cohérente des alias pour améliorer la compréhension.
- **Sécurité** : Paramètres dynamiques pour éviter les injections SQL.

L'ensemble de ces requêtes a été testé et validé sur la base de données. Le code SQL complet est disponible en annexe pour référence.

Conclusion

Le projet a permis de concevoir et de réaliser un système complet de gestion des péages, intégrant une modélisation claire des données, la création d'un schéma relationnel robuste, ainsi qu'une interaction fluide entre une base de données PostgreSQL et une application Java. Les principales fonctionnalités, comme la gestion des abonnements, la génération de tickets, et l'analyse des trajets, ont été mises en œuvre avec succès et testées efficacement.

Bilan des travaux réalisés :
Les étapes du projet, de l'analyse fonctionnelle à la traduction en code, ont permis de répondre aux objectifs initiaux. Le système repose sur un modèle bien structuré, garantissant la cohérence des données et leur accessibilité. Les choix techniques, tels que l'utilisation des requêtes SQL préparées et d'une application Java modulaire, renforcent la sécurité et la maintenabilité du projet.