Rapport de Projet : Gestion Automatisée de la Base de Données Velib'

Mimouna Nour elkamar

1 Décembre 2024

Table des matières

1 Introduction

Ce projet a pour objectif de concevoir un système de gestion de base de données pour le réseau Velib', en utilisant les données ouvertes disponibles et en intégrant des fonctionnalités avancées comme la génération de données, des triggers, et une visualisation interactive. Ce document décrit les étapes de développement, les algorithmes utilisés, ainsi que les résultats obtenus.

Le projet se compose de plusieurs étapes : le traitement des données, la génération de données simulées, l'intégration des triggers pour contrôler les tables, la planification des maintenances, et la visualisation des résultats.

2 Étapes Méthodologiques

2.1 Téléchargement et Traitement des Données

Les données des stations Velib' ont été récupérées depuis le fichier JSON disponible en ligne, fourni par le service Open Data Velib'. Ce fichier contient des informations essentielles, telles que :

- **Identifiant de la station** (`station_id`) : Un identifiant unique pour chaque station.
- **Nom de la station** (`name`) : Le nom de la station Velib'.
- **Coordonnées GPS** (`latitude`, `longitude`) : La position géographique de chaque station.
- **Statut de la station** (`status`) : Indique si la station est ouverte ou fermée.

Ces données ont été traitées via un script Python pour extraire les informations pertinentes et les convertir en instructions SQL. Cela permet de peupler la table `stations` de la base de données de manière structurée et d'assurer que les données sont cohérentes avec le modèle relationnel de la base.

2.2 Génération des Utilisateurs

Pour simuler un environnement d'utilisation réel, la table `utilisateurs` a été remplie avec des utilisateurs fictifs générés à l'aide de la bibliothèque Python `Faker`. Ce processus a permis de créer des profils d'utilisateurs avec les attributs suivants :

- **Nom complet** (`nom`) : Un nom généré aléatoirement pour chaque utilisateur.
- **Adresse email** (`email`) : Une adresse email unique pour chaque utilisateur.
- **Autres informations** : Comme des numéros de téléphone ou des adresses physiques, selon les besoins.

Ces utilisateurs ont ensuite été insérés dans la base via un fichier SQL généré par le script Python. Ce fichier `inserts_utilisateurs.sql` a permis de peupler la base de données avec des utilisateurs réalistes, offrant ainsi un cadre pour simuler l'interaction des clients avec le service Velib'.

2.3 Génération des Vélos

La table `velo` a été remplie en simulant des vélos avec des attributs aléatoires. Pour chaque vélo, les caractéristiques suivantes ont été générées :

— **Statut du vélo** (`statut`) : Le statut peut être `DISPONIBLE`, `INDISPONIBLE`, ou `TRAJET`, en fonction de critères comme le nombre de trajets effectués ou la note moyenne du vélo.
— **Nombre de trajets** (`nb_trajets`) : Un entier aléatoire entre 0 et 60, représentant le nombre de trajets effectués par le vélo.
— **Note moyenne** (`note_moyenne`) : Une note aléatoire entre 0 et 10, indiquant l'état général du vélo, calculée en fonction des retours des utilisateurs.

En fonction de la note moyenne ou du nombre de trajets, certains vélos sont automatiquement marqués comme `INDISPONIBLE`. Les vélos avec un statut `DISPONIBLE` ont été assignés aléatoirement à des stations. Ce script permet de peupler la base de données avec un nombre suffisant de vélos variés pour simuler un système de location de vélos en fonctionnement.

2.4 Triggers pour Contrôler les Tables

Des triggers ont été créés pour automatiser certaines opérations au sein de la base de données, réduisant ainsi les interventions manuelles. Un trigger principal a été mis en place pour chaque mise à jour du statut d'un vélo :

— Après chaque mise à jour, le trigger vérifie si le vélo est `DISPONIBLE`.
— Si le vélo a une note moyenne inférieure à un certain seuil ou un nombre de trajets trop élevé, il est automatiquement marqué comme `INDISPONIBLE`.
— Si le statut d'un vélo passe à `DISPONIBLE`, une tâche de maintenance est créée dans la table `maintenance` avec des informations telles que la gravité de la maintenance, l'identifiant du technicien affecté et les dates de début et de fin de l'intervention.

Ce trigger permet de garantir que la maintenance des vélos est bien gérée en fonction de leur état, sans nécessiter d'intervention manuelle.

2.5 Planification des Maintenances

La planification des maintenances a été réalisée en utilisant un algorithme de priorité. Les étapes de la planification sont les suivantes :

— Les vélos sont triés par gravité de maintenance, de la plus `CRITIQUE` à la plus `FAIBLE`.
— Un technicien est assigné à chaque vélo en fonction de la disponibilité et de la proximité géographique avec la station où se trouve le vélo.
— Le planning des maintenances est optimisé pour minimiser les déplacements des techniciens et assurer que les vélos les plus endommagés sont réparés en priorité.

Ce système de planification permet de maximiser l'efficacité des interventions et de garantir une gestion optimale des maintenances des vélos.

2.6 Visualisation des Données

Une carte interactive a été générée pour représenter visuellement le planning des maintenances et les stations concernées. Voici comment cette carte a été mise en place :

— Les coordonnées GPS de chaque station et les informations de maintenance sont extraites de la base de données.
— Un marqueur est placé sur la carte pour chaque station, avec des couleurs ou des icônes représentant le niveau de gravité de la maintenance à effectuer.

— Des trajets sont tracés entre les stations pour représenter l'ordre dans lequel les techniciens doivent se rendre sur chaque station pour effectuer la maintenance.

Cette carte permet de visualiser facilement l'état des stations et de suivre le travail des techniciens en temps réel. Elle facilite également la planification logistique des interventions.

3 Diagrammes

3.1 Diagrammes Tables

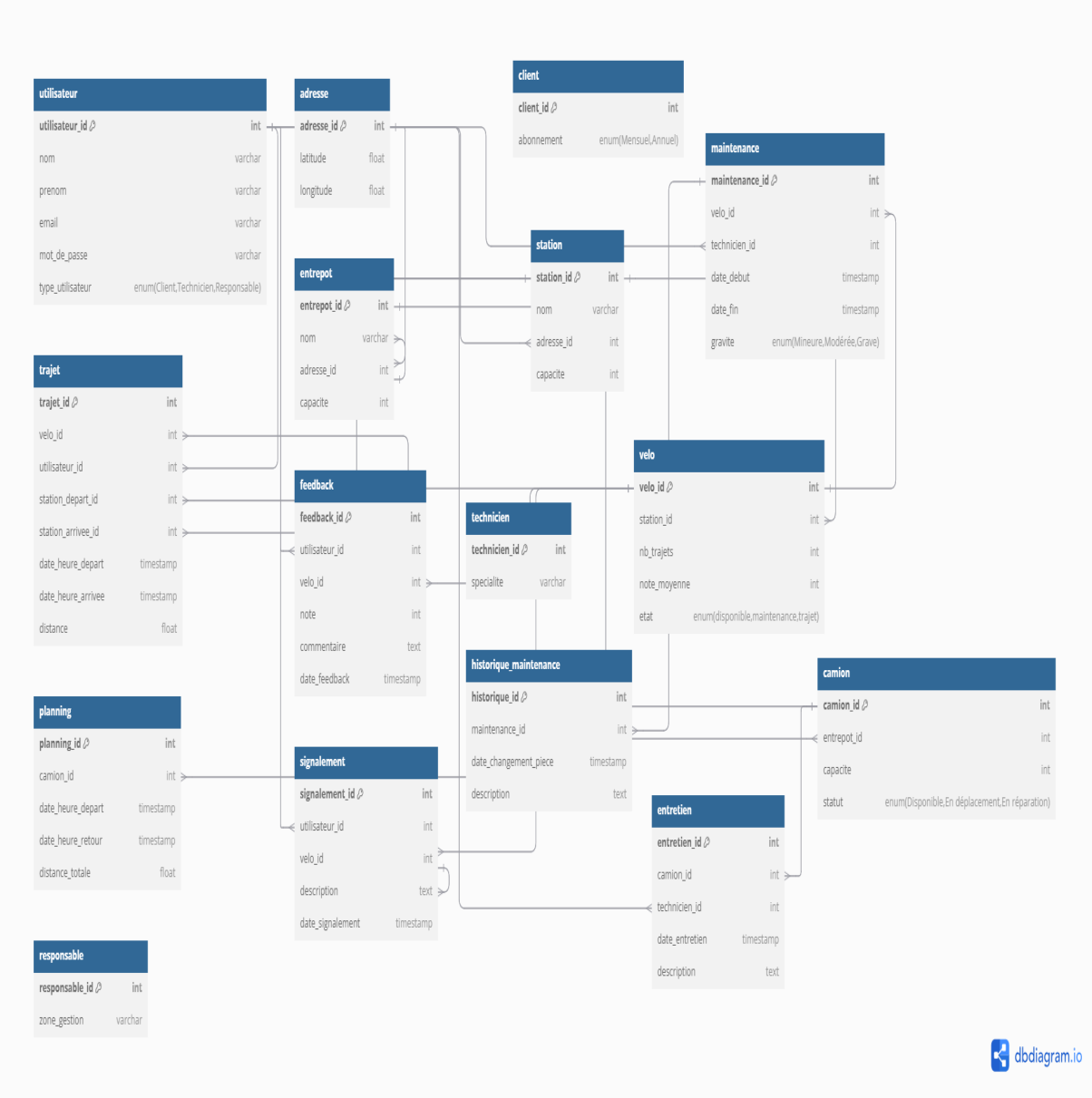

FIGURE 1 – Diagramme DES relations entre les tables

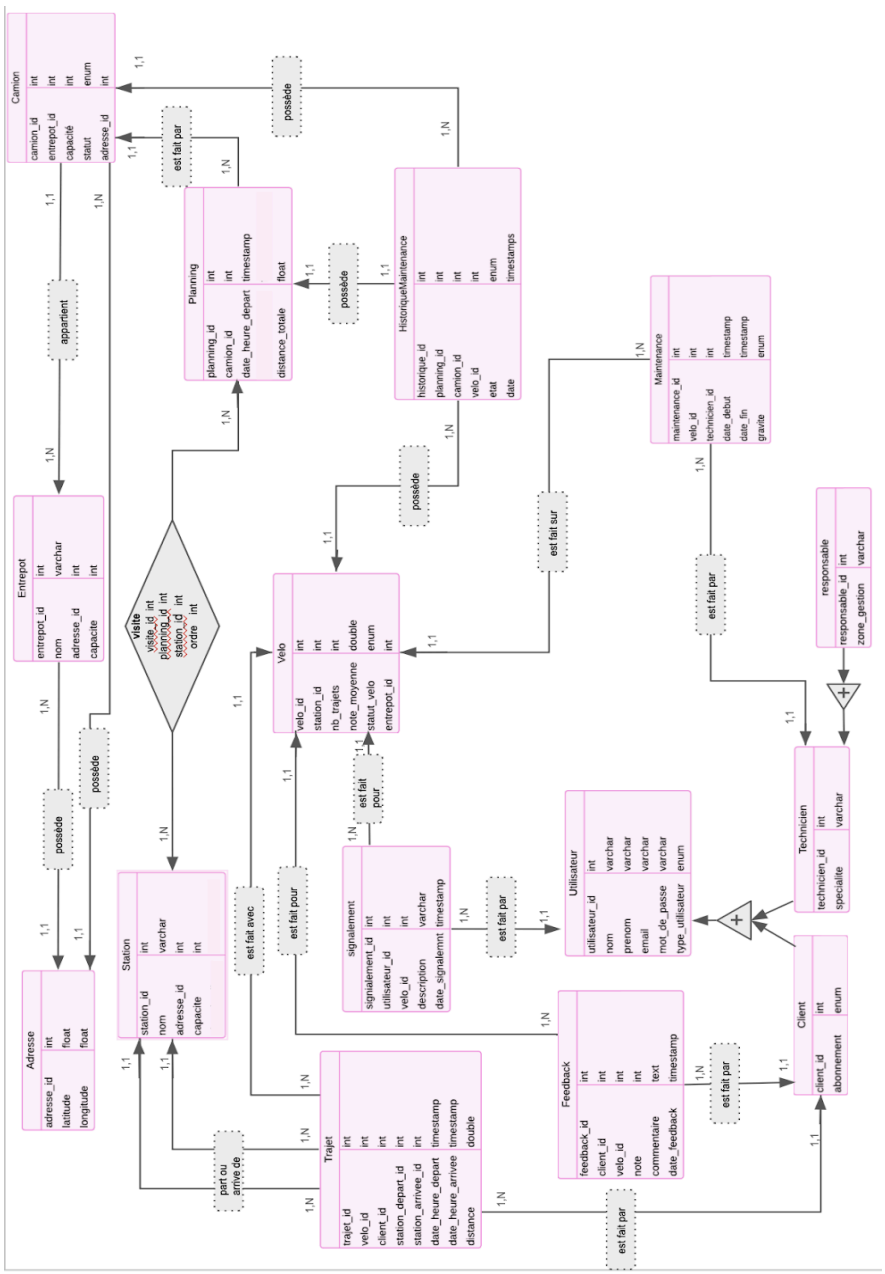

FIGURE 2 – Diagramme d'activité pour le trigger de maintenance

4 Tests et Automatisation

4.1 Scripts de Test

Un fichier `test.sql` a été conçu pour valider les fonctionnalités de la base de données. Il vérifie l'intégrité des données, l'exécution correcte des triggers, et la cohérence des relations entre les tables.

4.2 Automatisation avec Makefile

Un Makefile a été développé pour exécuter l'ensemble du projet, incluant le téléchargement des données, la génération des fichiers SQL, le peuplement de la base, et les tests.

5 Conclusion

Ce projet a permis de concevoir un système complet et automatisé pour la gestion des données Velib', intégrant des fonctionnalités avancées et des outils modernes. Les résultats obtenus démontrent une gestion efficace des données et un potentiel pour des applications réelles.

A Annexe : Code Source

A.1 Scripts SQL

A.2 Génération des Tables

```
1
2  -- Ce fichier me sert    creer mes tables et donc ajuster leurs
       c l  s    la fin du fichier
3
4  CREATE TYPE statut_camion AS ENUM ('DISPONIBLE', 'EN
       D PLACEMENT', 'EN R PARATION');
5  CREATE TYPE gravite_maintenance AS ENUM ('FAIBLE', 'MOYENNE', '
       CRITIQUE');
6  CREATE TYPE abonnement_client AS ENUM ('MENSUEL', 'ANNUEL');
7  CREATE TYPE type_utilisateur AS ENUM ('CLIENT', 'TECHNICIEN', '
       RESPONSABLE');
8  CREATE TYPE statut_velo AS ENUM ('DISPONIBLE', 'MAINTENANCE', '
       TRAJET', 'INDISPONIBLE');
9
10
11
12 -- Table Adresse
13 CREATE TABLE adresse (
14     adresse_id SERIAL PRIMARY KEY,
15     latitude DOUBLE PRECISION,
16     longitude DOUBLE PRECISION
17 );
18
19
20 -- Table Station
21 CREATE TABLE station (
22     station_id SERIAL PRIMARY KEY,
23     nom VARCHAR(100),
24     adresse_id INT REFERENCES adresse(adresse_id),
25     capacite INT
26 );
27
28 -- Table Entrep t
29 CREATE TABLE entrepot (
30     entrepot_id SERIAL PRIMARY KEY,
31     nom VARCHAR(100),
32     adresse_id INT REFERENCES adresse(adresse_id),
33     capacite INT
34 );
35
36 -- Table Camion
37 CREATE TABLE camion (
38     camion_id SERIAL PRIMARY KEY,
39     entrepot_id INT REFERENCES entrepot(entrepot_id),
40     capacite INT,
```

```
    statut statut_camion,
    adresse_id INT REFERENCES adresse(adresse_id)

);

-- Table Velo
CREATE TABLE velo (
    velo_id SERIAL PRIMARY KEY,
    station_id INT REFERENCES station(station_id),
    entrepot_id INT REFERENCES entrepot(entrepot_id),
    nb_trajets INT,
    note_moyenne DOUBLE PRECISION,
    statut statut_velo
);

-- Table Utilisateur
CREATE TABLE utilisateur (
    utilisateur_id SERIAL PRIMARY KEY,
    nom VARCHAR(100),
    prenom VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    mot_de_passe VARCHAR(255),
    type_utilisateur type_utilisateur
);

-- Table Technicien (hérite de Utilisateur)
CREATE TABLE technicien (
    technicien_id INT PRIMARY KEY REFERENCES utilisateur(utilisateur_id),
    specialite VARCHAR(100)
);

-- Table Client (hérite de Utilisateur)
CREATE TABLE client (
    client_id INT PRIMARY KEY REFERENCES utilisateur(utilisateur_id),
    abonnement abonnement_client
);

-- Table Responsable (hérite de Utilisateur)
CREATE TABLE responsable (
    responsable_id INT PRIMARY KEY REFERENCES utilisateur(utilisateur_id),
    zone_gestion VARCHAR(100)
);

-- Table Feedback
CREATE TABLE feedback (
    feedback_id SERIAL PRIMARY KEY,
    client_id INT REFERENCES client(client_id),
    velo_id INT REFERENCES velo(velo_id),
```

```
    note INT CHECK (note >= 1 AND note <= 5),
    commentaire TEXT,
    date_feedback TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table Trajet
CREATE TABLE trajet (
    trajet_id SERIAL PRIMARY KEY,
    velo_id INT REFERENCES velo(velo_id),
    client_id INT REFERENCES client(client_id),
    station_depart_id INT REFERENCES station(station_id),
    station_arrivee_id INT REFERENCES station(station_id),
    date_heure_depart TIMESTAMP,
    date_heure_arrivee TIMESTAMP,
    distance DOUBLE PRECISION
);

-- Table Maintenance
CREATE TABLE maintenance (
    maintenance_id SERIAL PRIMARY KEY,
    velo_id INT REFERENCES velo(velo_id),
    technicien_id INT REFERENCES technicien(technicien_id),
    date_debut TIMESTAMP,
    date_fin TIMESTAMP,
    gravite gravite_maintenance
);

-- Table Planning
CREATE TABLE planning (
    planning_id SERIAL PRIMARY KEY,
    camion_id INT REFERENCES camion(camion_id),
    date TIMESTAMP,
    distance_totale DOUBLE PRECISION
);

-- Table Signalement
CREATE TABLE signalement (
    signalement_id SERIAL PRIMARY KEY,
    utilisateur_id INT REFERENCES utilisateur(utilisateur_id),
    velo_id INT REFERENCES velo(velo_id),
    description TEXT,
    date_signalement TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table HistoriqueMaintenance
CREATE TABLE historiqueMaintenance (
    historique_id SERIAL PRIMARY KEY, -- Cl  primaire
    planning_id INT REFERENCES planning(planning_id) ON DELETE
   SET NULL,
    camion_id INT REFERENCES camion(camion_id) ON DELETE SET
   NULL, -- Permet de g rer la suppression
```

```
138     velo_id INT REFERENCES velo(velo_id) ON DELETE CASCADE, -- Suppression en cascade si le v lo est supprim
139     etat VARCHAR(10) CHECK (etat IN ('RECUP', 'RETOUR')), -- Contraintes pour limiter les valeurs possibles
140     date TIMESTAMP DEFAULT CURRENT_TIMESTAMP -- D finit une date par d faut si non fournie
141 );
142
143
144 -- Table Visite
145 CREATE TABLE visite (
146     planning_id INT NOT NULL REFERENCES planning(planning_id) ON DELETE CASCADE, -- Cl  trangre  avec suppression en cascade
147     station_id INT NOT NULL REFERENCES station(station_id) ON DELETE CASCADE,  -- Cl  trangre  avec suppression en cascade
148     ordre INT NOT NULL, -- Indique l'ordre de la visite dans le planning
149     PRIMARY KEY (planning_id, station_id) -- Cl  primaire compos e
150 );
151
152
153 ALTER TABLE station ALTER COLUMN station_id TYPE BIGINT;
154
155 -- Ajouter le statut dans planning
156 ALTER TABLE planning ADD COLUMN statut VARCHAR(20) DEFAULT 'EN_ATTENTE';
157
158 -- Ajouter le statut dans visite
159 ALTER TABLE visite ADD COLUMN statut VARCHAR(20) DEFAULT 'A_FAIRE';
160
161 ALTER TABLE velo ALTER COLUMN velo_id TYPE BIGINT;
162
163 ALTER TABLE station ALTER COLUMN station_id TYPE BIGINT;
164
165 INSERT INTO camion (camion_id, statut, capacite)
166 VALUES (1, 'DISPONIBLE', 100);
```

A.3 Génération des Triggers

```
1
2  --ce fichier sert a ajouter un trigger sur la table velo qui
       verifie la note du velo est ajoute une maintenance dans la
       table maintenance si la note du velo est inferieur    3
3
4  CREATE OR REPLACE FUNCTION trigger_ajouter_maintenance()
5  RETURNS TRIGGER AS $$
6  DECLARE
7      note_velo FLOAT;
8      gravite_maintenance gravite_maintenance;
9  BEGIN
10     -- R cup rer la note moyenne du v lo
11     SELECT note_moyenne INTO note_velo FROM velo WHERE velo_id
       = NEW.velo_id;
12
13     -- D terminer la gravit  de la maintenance
14     IF note_velo >= 5 THEN
15         gravite_maintenance := 'FAIBLE'::gravite_maintenance;
16     ELSIF note_velo >= 3 THEN
17         gravite_maintenance := 'MOYENNE'::gravite_maintenance;
18     ELSE
19         gravite_maintenance := 'CRITIQUE'::gravite_maintenance;
20     END IF;
21
22     -- Ins rer un enregistrement dans la table Maintenance
23     INSERT INTO maintenance (velo_id, technicien_id, date_debut
       , date_fin, gravite)
24     VALUES (
25         NEW.velo_id,
26         (SELECT technicien_id FROM technicien ORDER BY RANDOM()
       LIMIT 1), -- Technicien al atoire
27         CURRENT_TIMESTAMP - INTERVAL '1 DAY', -- Date de d but
       fictive
28         CURRENT_TIMESTAMP,                    -- Date de fin
       fictive
29         gravite_maintenance
30     );
31
32     RETURN NEW;
33 END;
34 $$ LANGUAGE plpgsql;
35
36 CREATE TRIGGER trigger_ajouter_maintenance
37 AFTER UPDATE ON velo
38 FOR EACH ROW
39 WHEN (NEW.statut = 'DISPONIBLE')
40 EXECUTE FUNCTION trigger_ajouter_maintenance();
```

```
1  -- ces triggers ils assurent les transitions des etats velo de
       indisponible    maintenance en ajoutant que le velo est
       recuper     la table historique maintenance avec la dat edu
        jour
2
3  CREATE OR REPLACE FUNCTION trigger_recuperer_velo_condition()
4  RETURNS TRIGGER AS $$
5  BEGIN
6      IF NEW.statut = 'FAIT' THEN
7          -- Ajouter une entr e dans historiqueMaintenance
8          INSERT INTO historiqueMaintenance (camion_id, velo_id,
       statut, date)
9          VALUES (NEW.planning_id, NEW.velo_id, 'RECUP',
       CURRENT_TIMESTAMP);
10
11         -- Mettre    jour les informations du v lo
12         UPDATE velo
13         SET statut = 'MAINTENANCE',
14             station_id = NULL,
15             entrepot_id = (SELECT entrepot_id
16                            FROM camion
17                            WHERE camion.camion_id = NEW.
       planning_id)
18         WHERE velo_id = NEW.velo_id;
19     END IF;
20
21     RETURN NEW;
22 END;
23 $$ LANGUAGE plpgsql;
24
25 CREATE TRIGGER trigger_recuperation_condition
26 AFTER UPDATE ON visite
27 FOR EACH ROW
28 WHEN (NEW.statut = 'FAIT')
29 EXECUTE FUNCTION trigger_recuperer_velo_condition();
```

```
1
2  -- cest triggers assurent que une fois qu'une nouvelle ligne
       sajoute dans signalement trajet et feedback verifie cele est
        mets    jour le statut du velo,
3  -- pour signalement on verifie que le commentaire n'a pas ces
       mots cle = bad|trous|ne marche pas|shit
4  --pour feedback on met a jour la note du velo et on voit si
       elle est <3
5  --pour trajet on verifie aussi la note du velo on mets a jour
       le nombre de trajet et on verifie qu'il nest pas < 40  et
       puis on met a jour la station su velo
6  CREATE OR REPLACE FUNCTION update_bike_status_on_signalement()
7  RETURNS TRIGGER AS $$
8  BEGIN
9      IF NEW.description ~* '(bad|trous|ne marche pas|shit)' THEN
10         UPDATE velo
11         SET etat = 'INDISPONIBLE'
12         WHERE velo_id = NEW.velo_id;
13     END IF;
14     RETURN NEW;
15 END;
16 $$ LANGUAGE plpgsql;
17
18 CREATE TRIGGER trg_update_bike_status_signalement
19 AFTER INSERT ON signalement
20 FOR EACH ROW
21 EXECUTE FUNCTION update_bike_status_on_signalement();
22
23
24
25 -- Fonction PL/pgSQL pour mettre    jour les statistiques du
       v lo apr s un feedback
26 -- Fonction PL/pgSQL pour mettre    jour les statistiques du
       v lo apr s un feedback
27 CREATE OR REPLACE FUNCTION update_bike_status_on_feedback()
28 RETURNS TRIGGER AS $$
29 DECLARE
30     current_nb_trajets INTEGER;
31     current_note_moyenne NUMERIC;
32 BEGIN
33     -- R cup rer les valeurs actuelles de nb_trajets et
       note_moyenne pour le v lo
34     SELECT velo.nb_trajets, velo.note_moyenne INTO
       current_nb_trajets, current_note_moyenne
35     FROM velo
36     WHERE velo_id = NEW.velo_id;
37
38     -- Si le v lo a d j  des trajets, mettre    jour les
       statistiques
39     IF current_nb_trajets > 0 THEN
```

```
40          UPDATE velo
41          SET
42              note_moyenne = (current_note_moyenne *
    current_nb_trajets + NEW.note) / (current_nb_trajets + 1)
43
44          WHERE velo_id = NEW.velo_id;
45      ELSE
46          -- Si le v lo n'a pas de trajets, initialiser les
    statistiques
47          UPDATE velo
48          SET
49              note_moyenne = NEW.note
50
51          WHERE velo_id = NEW.velo_id;
52      END IF;
53
54      -- V rifier si la note moyenne est inf rieure    3 pour
    changer l' tat
55      SELECT velo.note_moyenne INTO current_note_moyenne
56      FROM velo
57      WHERE velo_id = NEW.velo_id;
58
59      IF current_note_moyenne < 3 THEN
60          UPDATE velo
61          SET statut = 'INDISPONIBLE'
62          WHERE velo_id = NEW.velo_id;
63      END IF;
64
65      RETURN NEW;
66  END;
67  $$ LANGUAGE plpgsql;
68
69  -- Cr er le trigger associ     la table feedback
70  CREATE TRIGGER trg_update_bike_status_feedback
71  AFTER INSERT ON feedback
72  FOR EACH ROW
73  EXECUTE FUNCTION update_bike_status_on_feedback();
74
75
76  -- Supprimer le trigger existant
77  CREATE OR REPLACE FUNCTION update_trajet_and_note()
78  RETURNS TRIGGER AS $$
79  DECLARE
80      avg_note NUMERIC;
81  BEGIN
82      -- Calcul de la moyenne des notes pour le v lo
83      SELECT AVG(note)::NUMERIC INTO avg_note
84      FROM feedback
85      WHERE velo_id = NEW.velo_id;
86
87      -- Mise    jour du v lo avec le nombre de trajets et la
```

```
    note moyenne
88      UPDATE velo
89      SET
90          nb_trajets = nb_trajets + 1,
91          note_moyenne = COALESCE(avg_note, 0),
92          station_id = NEW.station_arrivee_id  -- Assurez-vous
    que "station_arrivee" est bien le nom correct
93      WHERE velo_id = NEW.velo_id;
94
95      -- V rification de la note moyenne et mise    jour de l'
    tat  du v lo
96      IF COALESCE(avg_note, 0) < 3 THEN
97          UPDATE velo
98          SET statut = 'INDISPONIBLE'
99          WHERE velo_id = NEW.velo_id;
100     END IF;
101
102     RETURN NEW;
103 END;
104 $$ LANGUAGE plpgsql;
105
106 -- Cr er    nouveau le trigger
107 CREATE TRIGGER trg_update_trajet_and_note
108 AFTER INSERT ON trajet
109 FOR EACH ROW
110 EXECUTE FUNCTION update_trajet_and_note();
```

A.4 Script Python : Remplissage des tables

```python
import json
#J'importe mon fichier json, que j'ai copi  coller du open data de velib de la ville de paris donc c'est des vrais stations de la ville de paris
# Ce fichier python srt    remplir  dans un premier temps ma tables adresse et ensuite station

# le resulat de ce script est inserts_station_adresse.sql comme on a besoin d'une adresse pour creer une station je le fais en meme temps

# Charger le fichier JSON
with open('station.json') as f:
    data = json.load(f)
    count = 0

# Ouvrir le fichier inserts_station_adresse.sql en mode criture
with open('inserts_station_adresse.sql', 'w') as sql_file:
    # Insertion des adresses et des stations
#ici je choisis de mettre 15 stations pour faciliter mon travail mais ceci peut etre ajuste juste en modifiant la condition sur count

    for station in data['data']['stations']:
        if count >= 15:
            break  # Stopper apr s count stations

        lat = station['lat']
        lon = station['lon']
        # Insertion de l'adresse
        sql_file.write(f"INSERT INTO adresse (latitude, longitude) VALUES ({lat}, {lon});\n")

        # Remplacer les apostrophes dans le nom de la station
        station_name = station['name'].replace("'", "''")

        # Utilisation de count comme adresse_id pour la station
        sql_file.write(f"INSERT INTO station (station_id, nom, adresse_id, capacite) VALUES ({count}, '{station_name}', {count+1}, {station['capacity']});\n")

        count += 1
#pour verifier que c'est bon je fais ce printf mais cest pas obligatoire :)
print("Le fichier 'inserts_station_adresse.sql' a   t    g  n  r   avec succ s.")
```

```python
import random
#Une fois que mes tables stations et adresse sont crees et remplie je peux remplir ma table velo
# ici je suppose que j'ai pas de velo en MAINTENANCE car c'est le remplissage initiale de ma table avant toute modification
# le resulat de ce script est inserts_velo.sql
# Nombre de v los    simuler

nombre_velos = 10

# Charger les IDs des stations depuis un fichier CSV
with open('stations.csv', 'r') as f:
    next(f)  # Sauter l'en-t te
    stations = [line.strip() for line in f.readlines()]  # Liste des stations

with open('entrepots.csv', 'r') as f:
    next(f)  # Sauter l'en-t te
    entrepots = [int(line.strip()) for line in f.readlines()]  # Liste des entrep ts

# Ouvrir le fichier SQL pour  crire  les instructions INSERT
with open('inserts_velo.sql', 'w') as file:
    for i in range(nombre_velos):
        velo_id = i + 1  # ID unique pour chaque v lo
        statut = random.choice(['DISPONIBLE', 'TRAJET', 'INDISPONIBLE'])  # Statut al atoire
        nb_trajets = random.randint(0, 60)  # Nombre de trajets (0    60)
        note_moyenne = round(random.uniform(0, 10), 1)  # Note moyenne (0    10)

        # Initialisation des IDs
        station_id = "NULL"
        entrepot_id = "NULL"
        if nb_trajets > 40 or note_moyenne < 3 :
            statut = 'INDISPONIBLE'

        # Logique pour d terminer la station ou l'entrep t
        if statut == 'DISPONIBLE':
            station_id = f"'{random.choice(stations)}'"  # Assigner    une station

        elif statut == 'INDISPONIBLE' or nb_trajets > 50 or note_moyenne < 3:
            station_id = f"'{random.choice(stations)}'"  # Assigner    un entrep t

        elif statut == 'TRAJET':
            # Aucun station_id ni entrepot_id pour les v los
```

```
    en trajet
41              station_id = "NULL"
42              entrepot_id = "NULL"
43
44          # R daction de l'instruction SQL
45          file.write(
46              f"INSERT INTO velo (velo_id, station_id, nb_trajets
    , note_moyenne, statut, entrepot_id) "
47              f"VALUES ({velo_id}, {station_id}, {nb_trajets}, {
    note_moyenne}, '{statut}', {entrepot_id});\n"
48          )
49
50  print("Fichier 'inserts_velo.sql' g n r  avec succ s!")
```

```
from faker import Faker
import random
#Ce fichier sert a remplir les tables utilisateurs clients, technicien et responsable est ecris ces requets dans inserts_user.sql
# Initialisation de la biblioth que Faker
fake = Faker('fr_FR')

# Nombre d'entr es pour chaque type d'utilisateur
num_clients = 10
num_responsables = 1
num_techniciens = 2

# Ouvrir le fichier SQL en mode  criture
with open('inserts_users.sql', 'w') as sql_file:
    # G n rer les clients
    for _ in range(num_clients):
        nom = fake.last_name().replace("'", "''")
        prenom = fake.first_name().replace("'", "''")
        email = fake.email().replace("'", "''")
        mot_de_passe = fake.password(length=10)
        abonnement = random.choice(['MENSUEL', 'ANNUEL'])

        # Requ tes SQL pour ins rer un utilisateur et un client
        sql_file.write(
            f"INSERT INTO utilisateur (nom, prenom, email, mot_de_passe, type_utilisateur) "
            f"VALUES ('{nom}', '{prenom}', '{email}', '{mot_de_passe}', 'CLIENT');\n"
        )
        sql_file.write(
            f"INSERT INTO client (client_id, abonnement) "
            f"VALUES (currval('utilisateur_utilisateur_id_seq'), '{abonnement}');\n"
        )

    # G n rer les responsables
    for _ in range(num_responsables):
        nom = fake.last_name().replace("'", "''")
        prenom = fake.first_name().replace("'", "''")
        email = fake.email().replace("'", "''")
        mot_de_passe = fake.password(length=10)
        zone_gestion = fake.city().replace("'", "''")

        # Requ tes SQL pour ins rer un utilisateur et un responsable
        sql_file.write(
            f"INSERT INTO utilisateur (nom, prenom, email, mot_de_passe, type_utilisateur) "
```

```
43              f"VALUES ('{nom}', '{prenom}', '{email}', '{
    mot_de_passe}', 'RESPONSABLE');\n"
44          )
45          sql_file.write(
46              f"INSERT INTO responsable (responsable_id,
    zone_gestion) "
47              f"VALUES (currval('utilisateur_utilisateur_id_seq')
    , '{zone_gestion}');\n"
48          )
49
50      # G n rer les techniciens
51      for _ in range(num_techniciens):
52          nom = fake.last_name().replace("'", "''")
53          prenom = fake.first_name().replace("'", "''")
54          email = fake.email().replace("'", "''")
55          mot_de_passe = fake.password(length=10)
56          specialite = random.choice(['REPARATION', 'ENTRETIEN',
    'INSPECTION'])
57
58          # Requ tes SQL pour ins rer un utilisateur et un
    technicien
59          sql_file.write(
60              f"INSERT INTO utilisateur (nom, prenom, email,
    mot_de_passe, type_utilisateur) "
61              f"VALUES ('{nom}', '{prenom}', '{email}', '{
    mot_de_passe}', 'TECHNICIEN');\n"
62          )
63          sql_file.write(
64              f"INSERT INTO technicien (technicien_id, specialite
    ) "
65              f"VALUES (currval('utilisateur_utilisateur_id_seq')
    , '{specialite}');\n"
66          )
67
68  print("Le fichier 'inserts_users.sql' a   t   g  n  r   avec
    succ s.")
```

A.5 Script Python : Visualisation Cartographique

```
import pandas as pd
import folium

# Charger les fichiers CSV
stations_adresse = pd.read_csv('stations_adresse.csv')
adresses_ordre = pd.read_csv('adresses_ordre.csv')
entrepots = pd.read_csv('entrepots_adresse.csv')

# Fusionner les donn es pour inclure les informations
    n cessaires
stations = pd.merge(adresses_ordre, stations_adresse[['
    station_id', 'nom', 'latitude', 'longitude']],
                    on='station_id', how='inner')

# Cr er une carte centr e sur une position moyenne (Paris
    comme exemple)
moyenne_latitude = stations['latitude'].mean() if not stations.
    empty else entrepots['latitude'].mean()
moyenne_longitude = stations['longitude'].mean() if not
    stations.empty else entrepots['longitude'].mean()
carte = folium.Map(location=[moyenne_latitude,
    moyenne_longitude], zoom_start=12)

# Ajouter un encadr  avec le titre "Planning" en rouge
folium.map.Marker(
    [moyenne_latitude, moyenne_longitude],
    icon=folium.DivIcon(html=f"""
        <div style="background-color:white; border: 2px solid
    black; padding: 5px; text-align: center; width: 150px;">
            <h4 style="color:red; margin: 0;">Planning</h4>
        </div>
    """)
).add_to(carte)

# Ajouter les stations    la carte
for _, station in stations.iterrows():
    popup_text = f"<b>{station['nom']}</b><br>Ordre: {station['
    ordre']}"
    folium.Marker(
        location=[station['latitude'], station['longitude']],
        popup=popup_text,
        icon=folium.Icon(color="blue", icon="info-sign")
    ).add_to(carte)

# Ajouter les entrep ts    la carte
for _, entrepot in entrepots.iterrows():
    popup_text = f"<b>{entrepot['nom']}</b>"
    folium.Marker(
        location=[entrepot['latitude'], entrepot['longitude']],
```

```
42          popup=popup_text,
43          icon=folium.Icon(color="red", icon="warehouse")
44      ).add_to(carte)
45
46  # Sauvegarder la carte dans un fichier HTML
47  carte.save('carte_planning.html')
48
49  print("La carte avec le titre encadr  a   t   g n r e et
       sauvegard e sous le nom 'carte_planning.html'.")
```

A.6 Test and Makefile :

```
1
2  --cets le fichier de teste de la base je genere un feedback et
       un trajet pour voir si cets mis a jour dans le velo
       correspondant
3  --j'ai essayer de creer uen sorte de communication avce le
       terminal pour simuler le teste sur le terminal
4
5  -- Message d'introduction
6  \echo '===== D but des tests ====='
7
8
9  --  tape  1 : G n rer des trajets pour un v lo
10 \echo 'Ma Base Velib Actuelle'
11
12 select * from station;
13 select * from utilisateur;
14 select * from velo;
15
16 SELECT * FROM velo WHERE velo_id = 1;
17
18
19 INSERT INTO trajet (velo_id, date_heure_depart,
       date_heure_arrivee, station_depart_id, station_arrivee_id)
20 VALUES (1, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP + INTERVAL '10
       minutes', 1, 2);
21
22
23 -- V rification apr s mise    jour du nombre de trajets
24 \echo 'Nombre de trajet du velo de id 1 apr s le trajet'
25
26 SELECT * FROM velo WHERE velo_id = 1;
27
28
29
30 --  tape  2 : Ajouter un feedback pour un v lo
31 \echo '== SIMULATION FEEDBACK =='
32
33 \echo 'velo de id 2 avant le feedback'
34
35 SELECT * FROM velo WHERE velo_id = 2;
36
37
38
39
40
41 INSERT INTO feedback (feedback_id, velo_id, client_id, note,
       commentaire, date_feedback)
42 VALUES (1, 2, 2, 5, 'PAS BIEN :( !', CURRENT_TIMESTAMP);
43
```

```
44
45 \echo 'velo de id 2 apr s le feedback'
46 SELECT * FROM velo WHERE velo_id = 2;
47
48
49
50 -- V rification des v los devenant indisponibles
51 \echo 'La table VELO :'
52 SELECT * FROM velo ;
53
54 --  tape  4 : Cr er un planning avec des visites
55 \echo '== Cr ation de planning et de visites =='
56 -- je genere un planning et je fixe son id pour les testes d'
      apres
57
58 SELECT generate_planning(1);
59
60 SELECT * FROM planning WHERE planning_id = 1;
61
62 SELECT * FROM visite WHERE planning_id = 1;
63
64
65 SELECT * FROM historiqueMaintenance ;
66
67
68 --  tape  5 : Remettre les v los en station
69 \echo '== Retour des v los aux stations =='
70
71 select retour_velo(1);
72
73
74
75 -- Retourner les v los en maintenance    une station
76 \echo '    Mise    jour des v los retourn s et disponibles...
      '
77
78
79 -- V rification apr s retour des v los
80 \echo 'V rification des v los remis en station :'
81 SELECT * FROM velo ;
82 SELECT * FROM historiqueMaintenance;
83
84 \echo '===== Fin des tests ====='
```

```
DB_NAME=velib_test
USER=postgres
HOST=localhost
PORT=5432
PGPASSWORD=NouSr2003

.PHONY: all test clean

all: reset setup python_prepare python_exports python_run
    inserts create_csv test create_1 python_carte

# ce make file est crucial dans le teste de ma base faut creer
    une base et modifier les informations en haut , nom de base
    , mot de passe
# l'ordre est notemment imoportant ici car les fichiers
    d pendent les uns des autres.
# a la fin on genere une caret avec les stations et l'ordre de
    visite

reset:
  PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
    ) -d $(DB_NAME) -c "DROP SCHEMA public CASCADE;"
  PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
    ) -d $(DB_NAME) -c "CREATE SCHEMA public;"
#  tape  1 : Ex cute les scripts SQL dans l'ordre pour cr er
    les tables et triggers
setup:
  PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
    ) -d $(DB_NAME) -f code1.sql
  PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
    ) -d $(DB_NAME) -f code2.sql
  PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
    ) -d $(DB_NAME) -f code3.sql
  PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
    ) -d $(DB_NAME) -f code4.sql
  PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
    ) -d $(DB_NAME) -f code5.sql

#  tape  2 : Ex cute code1.py pour g n rer
    inserts_station_adresse.sql
python_prepare:
  python3 code1.py

#  tape  3 : Exporte les donn es pour station et entrep t
    dans des fichiers CSV
python_exports:
  PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
    ) -d $(DB_NAME) -f inserts_station_adresse.sql
  PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
    ) -d $(DB_NAME) -f inserts_entrepot.sql
```

```
34    PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
      ) -d $(DB_NAME) -f inserts_velo.sql
35
36    PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
      ) -d $(DB_NAME) -c "\copy (SELECT station_id FROM station)
      TO 'stations.csv' CSV HEADER;"
37
38    PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
      ) -d $(DB_NAME) -c "\copy (SELECT entrepot_id FROM entrepot)
       TO 'entrepots.csv' CSV HEADER;"
39
40  #  tape  4 : Ex cute les scripts Python suivants pour
      g n rer d'autres donn es bas es sur les exports
41  python_run:
42    python3 code2.py
43    python3 code3.py
44
45  #  tape  5 : Ins re les autres donn es initiales
46  inserts:
47    PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
      ) -d $(DB_NAME) -f inserts_velo.sql
48    PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
      ) -d $(DB_NAME) -f inserts_users.sql
49
50  #  tape  6 : Tests et v rifications
51  test:
52    PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
      ) -d $(DB_NAME) -f test.sql
53
54  # Exporter les stations et adresses associ es    l'id_planning
       = 1
55  # Exporter les stations avec adresse, latitude, longitude et
      ordre pour planning_id = 1
56  create_csv :
57
58
59
60    PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
      ) -d $(DB_NAME) -c "\copy (SELECT e.entrepot_id, e.nom, a.
      adresse_id, a.latitude, a.longitude FROM entrepot e JOIN
      adresse a ON e.adresse_id = a.adresse_id) TO '
      entrepots_adresse.csv' DELIMITER ',' CSV HEADER;"
61    PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
      ) -d $(DB_NAME) -c "\copy (SELECT s.station_id, s.nom, a.
      adresse_id, a.latitude, a.longitude FROM station s JOIN
      adresse a ON s.adresse_id = a.adresse_id) TO '
      stations_adresse.csv' DELIMITER ',' CSV HEADER;"
62  create_1 :
63    PGPASSWORD=$(PGPASSWORD) psql -U $(USER) -h $(HOST) -p $(PORT
      ) -d $(DB_NAME) -c "\copy (SELECT a.station_id, v.ordre FROM
       station a JOIN visite v ON a.station_id = v.station_id
```

```
    WHERE v.planning_id = 1 ) TO 'adresses_ordre.csv' DELIMITER
    ',' CSV HEADER;"
64
65
66
67
68
69 python_carte :
70   python3 carte.py
71 # Nettoyage des fichiers CSV  g  n  r  s
72 clean:
73   rm -f stations.csv entrepots.csv inserts_station_adresse.sql
     inserts_velo.sql inserts_users.sql entrepots_adresse.csv
     adresses_ordre.csv stations_adresse.csv carte_planning.html
```