Gestion des parkings

Une société de parkings souhaite mettre en place une base de données afin de gérer ses différents parkings dans une ville. Chaque parking est composé de plusieurs places, en plein air ou couvertes, adaptées à certains types de véhicules (deux_roues; camion; véhicule simple). Les prix des parkings varient selon la zone dans laquelle ils se situent dans la ville (Centre-ville, zone industrielle, zone d'activités commerciales, etc.).

Chaque utilisateur de ces parkings est soit un occasionnel, soit un abonné chez cette société. Un utilisateur peut avoir un ou plusieurs véhicules. Les occasionnels paient des tickets à l'heure, dans des guichets ou sur des automates avec sa carte bancaire. Les abonnés disposent des cartes mensuelles renouvelables chaque mois. La société souhaite gérer les transactions ayant lieu lors de chaque paiement.

Pour attirer plus de clients, la société veut mettre en œuvre un système de fidélité pour ses clients. Pour chaque nouvel abonné, un compte lui est créé afin de sauvegarder ses différents abonnements.

Note de clarification :

1. **Parking :**
    a. adresse : string { key }
    b. est dans la zone => zone

    On calcule nombre de place total ainsi que le nombre de place occupé et libre et on l'affiche publiquement.
2. **Zone :**
    a. nom { key }
    b. prix : int
3. **Place :**
    a. numéro : int { key }
    b. est dans parking => parking
    c. emplacement: enum { plein air, couverte }
    d. type vehicule : enum { voiture, deux_roues, camion }

e. Status : enum(libre || occupé || reservé)

4. **Reservations :**
    a. **Véhicule => Véhicule**
    b. **place => place**
    c. **Prix**
    d. **Date de début**
    e. **Date de fin**

5. **Utilisateur :**
    a. Nom : string
    b. Prénom : string

    4.1. UtilisateurOccasionnel
    Herite de Utilisateur
        a. possède un véhicule => Véhicule

    4.2. UtilisateurAbonné
    Herite de Utilisateur
        a. num_abonné : int { key }
        b. possède un ou **plusieurs** véhicule => Véhicule
        c. Reduction : int { key }
        d. compte_actif : bool { key }
        e. Carte de paiement

6. **Vehicule :**
    a. num_immat : string { key }
    b. véhicule appartient à un utilisateur => Utilisateur
    c. type_vehicule : enum { voiture, deux_roues, camion }

7. **Transaction**
    a. Date de paiement
    b. Moyen de paiement : (CB || Cash)
    c. Machine : (Automate || Guichet || En ligne)
    La transaction est payée seulement en CB si elle a un lien sur un automate. La transaction est payable seulement sur un guichet ou en ligne si c'est un abonnement). La

transaction d’un utilisateur occasionnel est payable seulement en guichet ou automate.

6.1. TransactionOccasionnel :
Herite de Transaction

a. Prix de la zone => Zone

6.2. TransactionAbonné :
Herite de Transaction

b. PrixAbonnement : int { key }

8. **Ticket :**
    a. heure_arrivée : TIME { key }
    b. num_immat => véhicule
    c. place => place

Le système fonctionnerait de la manière suivante :

Un véhicule arrive à la barrière d’entrée, une caméra filme le véhicule et reconnaît la plaque d’immatriculation. Ensuite l’utilisateur demande un ticket sur l’automate, la date et l’heure d’entrée sont précisées. Si le véhicule appartient à un utilisateur abonné ou un utilisateur ayant fait une réservation, la barrière s’ouvre directement sans ticket.
Si le parking est plein, la barriere ne s’ouvrira pas.

Au moment de la sortie, l’utilisateur présente son ticket, et paye à l’heure. Si l’utilisateur est abonné, il sortira directement sans ticket. Toutes les informations de l'utilisateur abonné (entrée et sortie) seront disponibles en ligne ou sur un guichet. Si l’utilisateur a fait une réservation et n’a pas dépassé le temps compris, la barrière s’ouvre, sinon il paye.