Étude de cas informatique

Préliminaires

L'énoncé proposé s'inspire d'un projet concernant des livraisons par des véhicules. Chaque partie comprend la définition d'un certain nombre de problématiques à résoudre et présente des objectifs concrets, ainsi que la réflexion sur les moyens de les atteindre.

Attendus. Il est attendu des candidates et des candidats des réponses construites. Ils seront aussi évalués sur la précision, le soin et la clarté de la rédaction.

Dépendances. Ce sujet contient cinq parties. Les différentes parties et un grand nombre de leurs questions sont largement indépendantes. Il est possible d'aborder les différentes parties dans l'ordre qui vous conviendra le mieux mais en indiquant clairement quelle question est répondue.

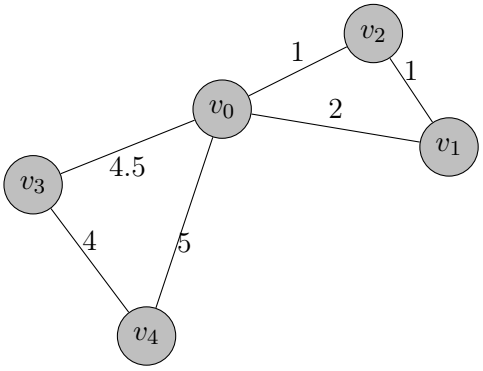
Partie I. Déterminer les tournées de véhicules

Dans cette partie, nous allons nous intéresser au problème de tournée de véhicules : un entrepôt stocke des produits qui doivent être livrés à des clients. Chaque client commande un nombre de produits (qui peuvent être différents). Les produits sont livrés par véhicule, par exemple un camion, et le but est de trouver des tournées de livraison qui minimisent la distance totale de kilomètres parcourus par les véhicules.

Ce problème est NP-complet (avec un véhicule seulement, c'est déjà le problème du voyageur de commerce) et nous allons donc nous intéresser à des heuristiques.

Nous allons utiliser la représentation via un graphe non-orienté $G = (V, E)$ avec les sommets représentant les différentes implantations des clients et les arêtes représentent les chemins entre les clients. L'entrepôt se situe au sommet v_0 et les n sommets restants représentent les clients. On considère que tous les sommets sont accessibles à partir de v_0 (donc le graphe est connexe). De plus, les arêtes e_{ij} sont pondérées par d_{ij}, la distance entre v_i et v_j. Le nombre maximum de véhicules est n, ce qui serait la situation dans laquelle exactement un véhicule est associé à chaque destination de livraison (client). Chaque véhicule a une capacité maximale C de produits qu'il peut charger en nombre de palettes. Pour finir, chaque client v_i, $i \geq 1$, a une demande w_i de produits. Un exemple est donné dans la figure 1 où on voit le graphe d'un réseau routier (figure 1(a) avec l'entrepôt au milieu et 4 clients) et sa représentation sous forme de matrice d'adjacence (figure 1(c)) ainsi que les demandes des clients (table 1(b) dans la figure 1).

Pour commencer, nous allons calculer les plus courtes distances entre tous les sommets pour un graphe donné. Ceci peut être fait avec l'**algorithme de Floyd-Warshall**. L'algorithme calcule, pour chaque paire de sommets, la distance minimale parmi tous les chemins entre ces



(a) Carte

client	v_1	v_2	v_3	v_4
demande	5	6	4	3

(b) Demandes en nombre de palettes

$$A = \begin{pmatrix} 0 & 2 & 1 & 4{,}5 & 5 \\ 2 & 0 & 1 & \infty & \infty \\ 1 & 1 & 0 & \infty & \infty \\ 4{,}5 & \infty & \infty & 0 & 4 \\ 5 & \infty & \infty & 4 & 0 \end{pmatrix}$$

(c) Matrice d'adjacence

$$dist = \begin{pmatrix} 0 & 2 & 1 & 4{,}5 & 5 \\ 2 & 0 & 1 & 6{,}5 & 7 \\ 1 & 1 & 0 & 5{,}5 & 6 \\ 4{,}5 & 6{,}5 & 5{,}5 & 0 & 4 \\ 5 & 7 & 6 & 4 & 0 \end{pmatrix}$$

(d) Matrice des distances

FIGURE 1 – Exemple d'un graphe et sa représentation sous forme de matrice d'adjacence et sous forme de matrice des distances.

deux sommets. Le pseudo-code est donné dans l'algorithme 1. La matrice des distances produit par l'algorithme 1 en prenant la matrice d'adjacence de la figure 1(c)) en entrée, est donnée dans la figure 1(d)).

Question 1. Proposer une implémentation en Python de l'algorithme Floyd-Warshall présenté dans l'algorithme 1. La fonction prend la matrice A d'adjacence pondérée d'un réseau routier et produit une matrice *dist* des plus courtes distances entre tous les sommets. On considère pouvoir utiliser `np.inf` comme infini.

$dist \leftarrow A$
foreach vertex $z \in V$ **do**
  foreach vertex $x \in V$ **do**
    foreach vertex $y \in V$ **do**
      if $dist(x,z) \neq \infty$ & $dist(z,y) \neq \infty$ & $dist(x,z) + dist(z,y) < dist(x,y)$ **then**
        $dist(x,y) \leftarrow dist(x,z) + dist(z,y)$
return *dist*

Algorithme 1 : Floyd-Warshall.

Maintenant que nous savons créer la matrice des plus courtes distances à partir d'une matrice d'adjacence, dans la suite du sujet nous allons considérer directement des matrices de distances en entrée des algorithmes.

Nous allons aborder le problème des tournées des véhicules. Nous supposons que la demande de chaque client est inférieure à la capacité des véhicules, c'est-à-dire un client peut être livré en une fois par un véhicule.

Une première approche pour créer des tournées de véhicules peut être la suivante (**algorithme naïf**) : étant donnée la matrice des distances les plus courtes entre tous les points de la carte produit par l'algorithme Floyd-Warshall, on prend un premier véhicule et on lui affecte le client le plus proche. Puis, depuis ce client, on rajoute le client le plus proche de lui qui n'est pas encore affecté à la tournée si la capacité de chargement du véhicule n'est pas dépassée par cette affectation. On continue ainsi jusqu'à ce que la capacité restante du véhicule soit trop petite pour livrer en entier le client qu'on souhaite rajouter à la tournée. Dans ce cas, on commence une nouvelle tournée avec un autre véhicule selon le même principe jusqu'à ce que tous les clients soient affectés à des tournées. Chaque véhicule rentre à l'entrepôt à la fin de sa tournée.

Question 2. On considère la carte de la figure 1(a) et sa matrice des plus courtes distances (figure 1(d)). Chaque véhicule peut charger C=12 palettes. Les demandes de palettes des clients sont données dans la table 1(b) de la figure 1.

1. Appliquer l'**algorithme naïf** expliqué au dessus et donner les tournées avec leurs coûts en termes de nombre de kilomètres pour l'ensemble des véhicules. Le coût d'une tournée inclut le retour du véhicule à l'entrepôt via le plus court chemin.
2. En considérant l'objectif de minimiser la distance totale des kilomètres parcourus par tous les véhicules, peut-on trouver une meilleure solution ? Si oui, quelle est-elle ?

Question 3. Est-ce que la suppression du client v_2 change quelque chose pour la répartition des clients restants en tournées ? Expliquer.

Question 4. Implémenter en Python l'**algorithme naïf** via la fonction `AlgoNaif(dist, demandes, C)`. L'algorithme prend en entrée une matrice de distances entre tous les clients et l'entrepôt, les demandes des clients sous forme d'une liste ainsi que la capacité C (constante entière) des véhicules à disposition.

Question 5. Il est possible de trouver des situations dans lesquelles l'algorithme n'est pas optimal concernant notre critère de la distance totale des tournées. Donner deux configurations pour lesquelles l'algorithme se fait piéger. Un ou plusieurs dessins sont attendus.

Nous allons maintenant regarder une approche plus sophistiquée, l'**algorithme de Clarke et Wright** (*Savings algorithm*), dont le principe est le suivant :

1. On commence avec n tournées : $v_0 \to v_i \to v_0$, pour tout $i \geq 1$, c'est-à-dire chaque client est livré par un véhicule différent.
2. Calculer ensuite les économies $s(i, j)$ (*savings*) pour fusionner deux clients v_i et v_j en une même tournée : $s(i, j) = dist(i, 0) + dist(0, j) - dist(i, j)$, pour tout $i, j \geq 1$ et $i \neq j$;
3. Trier les économies par ordre décroissant ;

Tournez la page S.V.P.

4. Commencer en tête de la liste (restante) des économies, fusionner les deux tournées associées avec l'économie (restante) la plus élevée, si :

   (a) Les deux clients ne sont pas déjà dans la même tournée ;

   (b) Aucun des deux clients n'est à l'intérieur de sa tournée : Les deux clients sont connectés directement à l'entrepôt dans leur tournée respective et donc sont livrés en premier ou en dernier dans leur tournée respective ;

   (c) La somme des demandes des deux tournées à fusionner ne dépasse pas la capacité maximum du véhicule.

5. Répéter l'étape 4 jusqu'à ce que plus aucune économie ne puisse être faite.

Nous allons utiliser une classe `Tournee` qui représente la tournée d'un véhicule et qui propose les méthodes suivantes :

- `distance(self, distance)` : qui renvoie la longueur actuelle de la tournée,
- `fusionnable(self, autreTournee, clients, capacite)` : teste si la tournée peut être fusionnée avec `autreTournee`. Elle teste notamment les différents cas du point (2) de l'**algorithme de Clarke et Wright**.
- `fusion(self, autreTournee, clients)` : qui prend en entrée la tournée avec laquelle elle doit faire la fusion et une liste de clients par lesquels la tournée fusionnée doit être reliée.
- `affiche(self)` : affiche la tournée avec la liste des différents clients à parcourir et le chargement total de cette tournée.

Le listing 1 donne des exemples d'appel.

Listing 1 – Code d'utilisation de la classe `Tournee`.

```
tournee1.affiche() # tournee 0 2 3 4 0 chargement 8
tournee2 = Tournee(1,3) #client 1 demande 3 palettes
tournee1.fusionnable(tournee2, [3,1], 12) #False
tournee1.fusionnable(tournee2, [2,1], 12) #True
tournee1.fusion(tournee2, [2,1])
tournee1.affiche() # tournee 0 4 3 2 1 0, chargement 11
# le parcours 0-4-3-2-1-0 est equivalent a 0-1-2-3-4-0
```

Question 6. Implémenter la classe `Tournee` en Python.

Question 7. En utilisant la classe `Tournee`, implémenter en Python l'**algorithme de Clarke et Wright** via la fonction `ClarkeWright(dist, demandes, C)`. L'algorithme prend en entrée la matrice des distances des plus courts chemins, les demandes des clients et la capacité des véhicules. Il renvoie la liste des tournées ainsi que la longueur totale des tournées.

Question 8. Il peut être nécessaire de livrer certaines clients en urgence. Discuter (sans faire l'implémentation dans le code) des modifications de l'algorithme précédent nécessaires pour prendre en compte des priorités dans la liste des clients.

La version actuelle du problème de tournée de véhicules fait abstraction d'un certain nombre de contraintes supplémentaires qui font partie du cas réel, telles que le temps de conduite des conducteurs, la date de contrôle technique des véhicules, leur capacité ou leur plaque d'immatriculation.

On veut maintenant tenir compte de la différence de caractéristiques des différents véhicules. On considère avoir une classe `Vehicule` dont une instance sera associée à chaque tournée. Un véhicule sera associé à une tournée lors de la création de cette dernière. Dans les trois questions suivantes, on se limitera à considérer qu'un véhicule ne possède comme attributs que sa capacité en nombre de palettes (un entier) et une plaque d'immatriculation (une chaîne de caractères). On veut modifier la classe `Tournee` pour intégrer les véhicules, sans modifier la liste des méthodes de cette classe.

Question 9. Est-il nécessaire de modifier les arguments des méthodes de `Tournee` ? Si oui, donner et expliciter les prototypes des méthodes ainsi modifiées ou ajoutées, *i.e.* donnez leur nom, leurs arguments, ainsi que la sémantique des arguments ajoutés, enlevés ou modifiés.

Question 10. Est-il nécessaire de modifier le code des méthodes de `Tournee` ? Si oui, lesquelles (sans donner le code ainsi modifié) ?

Certaines informations concernant les véhicules (la plaque d'immatriculation par exemple) sont utiles pour l'exécution des tournées, mais ne sont pas utile dans les algorithmes précédents. Pour que l'équipe de développement en charge de programmer `Tournee` et celle en charge de programmer `Vehicule` travaillent efficacement, on définit une classe `Transport` qui se limite à garantir l'accès à la capacité. Les informations plus précises sur chaque véhicule resteront quand à elles dans `Vehicule`.

Question 11. En se limitant à la capacité (`capacite`) et au numéro d'immatriculation (`immatriculation`), proposer les classes `Transport` et `Vehicule` en vous limitant à leur définition (ligne commençant par `class`) et à leur constructeur.

Partie II. Prise en compte du réseau

Lors de la tournée elle-même, les véhicules sont en communication constante avec le service central de coordination de la flotte. Cette communication réseau utilise des applications présentes dans chaque véhicule qui envoient régulièrement des données (vitesse, position...) au service central. Les communications entre les véhicules et le serveur peuvent être compliquées : zones de non couverture, tunnels, conditions météo...

Question 12. On suppose le cas où un véhicule se trouve bloqué dans un tunnel empêchant la communication. Que se passe-t-il (couche transport du point de vue de l'application embarquée) si on suppose que le véhicule veut envoyer une information au serveur en tentant d'établir une communication basée sur TCP ?

Tournez la page S.V.P.

Un blocage long dans une zone non connectée peut être lié à un embouteillage. Dans ce cas-là, la densité de véhicules peut permettre de transmettre des informations en utilisant un principe de communication inter-véhicules. On suppose pour les questions suivantes que tous les véhicules sont équipés d'un système de communication local sans fil compatible.

On teste un premier protocole de routage. Quand le véhicule veut envoyer une donnée au serveur, il l'envoie (`broadcast`) à tous les véhicules autour de lui. Ces derniers font de même sauf s'ils sont capables de contacter l'extérieur (véhicule de *bordure*). Dans ce cas-là, ils envoient le message au serveur.

Question 13. Ce protocole a un défaut fondamental, quel est-il ? Comment le corriger ?

Une fois ce problème corrigé, on se pose la question de la réception au niveau du serveur. On suppose avoir utilisé le protocole TCP entre les véhicules de *bordure* et le serveur. Si plusieurs véhicules sont en bordure, plusieurs messages applicatifs vont donc être envoyés vers le serveur pour la même donnée.

Question 14. Est-il nécessaire, au niveau de l'application recevant les données sur le serveur, de gérer ces multiples messages contenant des données identiques, ou est-ce déjà géré au niveau TCP ? Justifier votre réponse.

Lorsque la densité de véhicules est forte (comme lors d'un embouteillage), la distance entre les véhicules est faible par rapport à la portée de la communication sans fil. Lors d'un embouteillage on peut avoir plusieurs dizaines de véhicules à portée de communication directe. Dans un premier temps, on considère qu'un seul véhicule tente d'envoyer un message vers le serveur extérieur.

Question 15. Dans le cadre où un seul véhicule veut envoyer un message, tous les messages intermédiaires entre le véhicule initial et les véhicules de *bordure* sont-ils nécessaires ? Justifier.

Question 16. Lors d'une communication sans fil, il est possible d'obtenir au niveau applicatif la qualité du signal portant un message. Plus la distance entre l'émetteur et le récepteur est grande, plus le message est faible. Proposez un algorithme de retransmission de message permettant d'optimiser le nombre de messages utilisés.

Question 17. Discuter l'impact de cet algorithme en terme de nombre de messages et de latence. On pourra fixer par exemple le nombre de véhicule à distance de communication sans fil comme étant une constante.

Après des tests dans les situations d'embouteillages dans des tunnels, on réalise que plusieurs véhicules peuvent vouloir transmettre de l'information vers le serveur extérieur. Le nombre de messages émis pour être retransmis entre les véhicules est proportionnel au nombre de véhicules.

Le premier problème est lié à la nature des communications sans-fil. Si deux véhicules tentent de communiquer en même temps, les deux messages vont être brouillés.

Question 18. À quel couche du modèle OSI se pose ce premier problème ? Proposer une solution pour régler ce problème.

Le second problème est lié au nombre de messages de contenus différents.

Question 19. Tous les messages sont retransmis de proche en proche vers la bordure. Comment réduire le nombre de messages lorsqu'un grand nombre de véhicules tentent chacun d'envoyer indépendamment des données vers un serveur extérieur ?

Partie III. Archivage de la position, de la vitesse

Pour respecter la loi, les véhicules professionnels doivent conserver des informations légales, principalement la vitesse, dans un tachygraphe. On décide de mettre en place un tel tachygraphe numérique. On profite de sa mise en place pour ajouter une fonctionnalité. Les données sont envoyées au serveur en temps réel quand le véhicule est joignable, mais les données sont stockées localement dans le cas contraire. Pour garantir les aspects réglementaires (non modification des données par le personnel de l'entreprise), on utilise une machine virtuelle qui peut être déplacée entre le serveur et le véhicule lorsque la qualité de la communication entre ces deux éléments se dégrade. Chaque véhicule est suivi par une application tachygraphe qui se trouve dans une machine virtuelle spécifique à ce véhicule.

On considère que le serveur et l'ordinateur de bord sont d'architectures différentes (par exemple d'architecture x86 pour les serveurs et de type ARM pour les véhicules).

Question 20. Est-il possible de migrer l'application tachygraphe entre le serveur et le véhicule ? Si oui, expliciter l'impact sur les performances.

Cette application de tachygraphe permet de centraliser l'information pour plusieurs usages : archivage légal, mais aussi information du centre de coordination des véhicules, affichage pour le conducteur... Comme l'accès et la modification des données est complexe, elle peut prendre du temps. Aussi, il faut gérer en interne la synchronisation d'accès aux données pour plusieurs processus fournissant les services pour ces différents usages.

On suppose avoir deux fonctions `lecture(...)` et `ecriture(...)`, déjà implémentées, permettant respectivement de lire et d'écrire ces données qui peuvent prendre du temps. On veut avoir les propriétés suivantes :

1. Plusieurs lectures peuvent avoir lieu en même temps,
2. Les écritures sont exclusives (sans lecture ni écriture en même temps),
3. En cas de choix, priorité aux lecteurs.

Tournez la page S.V.P.

Un premier test est le suivant :

Listing 2 – code de la classe `Acces`.

```
import threading

class Acces:
    def __init__(self):
        self.mutex = threading.Semaphore(value = 1)

    def debut_lecture():
        self.mutex.acquire() # P
    def fin_lecture():
        self.mutex.release() # V

    def debut_ecriture():
        self.mutex.acquire()
    def fin_ecriture():
        self.mutex.release()
```

Avec une utilisation type (en considérant la variable `synchro` de classe `Acces`) :

Listing 3 – code d'utilisation de la classe `Acces`.

```
# Pour une lecture
...
synchro.debut_lecture()
valeur = lecture(...)
synchro.fin_lecture()
...

# Pour une ecriture
...
synchro.debut_ecriture()
ecriture(...)
synchro.fin_ecriture()
...
```

Question 21. Pour chacune des trois propriétés, expliquer pourquoi elle est respectée ou pourquoi elle ne l'est pas.

Question 22. Proposer (en Python) une classe `Acces_v2` permettant d'implémenter l'ensemble des propriétés souhaitées en conservant les mêmes méthodes.

Question 23. Donner un exemple de famine dans le cadre des propriétés souhaitées.

Partie IV. Gestion de tournées

Voici en figure 2 le schéma relationnel d'une base de données qui permet à l'entreprise de faire la gestion de ses tournées. La spécification des domaines est donnée dans le listing 4.

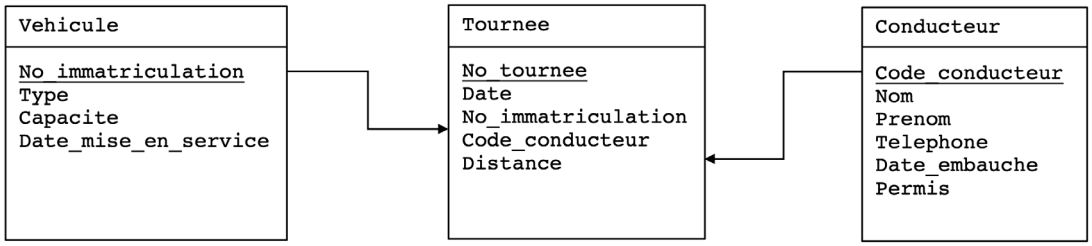


FIGURE 2 – Schéma relationnel de la base de donnée **Gestion de tournées**.

Listing 4 – Spécification de la base de données.

```
VEHICULE(*No_immatriculation(char(10)), Type(text), Capacite(float),
Date_mise_en_service(date))

TOURNEE(*No_tournee(int), Date(date), No_immatriculation(char(10)),
Code_conducteur(char(8)), Distance(float))

CONDUCTEUR(*Code_conducteur(char(8)), Nom(char(30)),
Prenom(char(40)), Telephone(texte), Date_embauche(date),
Permis(texte))
```

Question 24. Écrire une requête en SQL qui permet d'afficher combien de conducteurs ont un permis C1E (élément `Permis` dans la table précédente).

Question 25. Écrire une requête en SQL qui affiche la distance totale parcourue par John Doe en novembre 2021. On rappelle que les dates sont exprimées par défaut sous la forme `YYYY-MM-DD` dans les requêtes.

Question 26. Écrire une requête en SQL qui permet de calculer la capacité cumulée des véhicules selon le permis de leur conducteur.

Les différents véhicules nécessitent des permis de conduire différents. En effet, selon leur poids total autorisé en charge (PTAC), ou selon la présence éventuelle d'un attelage de remorque, il faut avoir le permis correspondant. On trouve un résumé des différents permis de transport de marchandises dans la table 1 et la figure 3. Le permis C1 permet ainsi de conduire un véhicule isolé dont le PTAC est compris entre 3,5t et 7,5t, tandis que le permis C1E permet de conduire un ensemble de véhicules dont le tracteur appartient à la catégorie C1 et dont le poids total ne dépasse pas 12t.

Dans sa forme actuelle, la base de données ne permet pas d'éviter les erreurs d'affectation. Il est pour l'instant tout à fait possible d'affecter un conducteur de moins de 21 ans à un véhicule isolé de plus de 7,5t de PTAC. Pour empêcher ce type d'erreur, l'entreprise souhaite améliorer la base de données en modifiant le schéma de manière à pouvoir effectuer une vérification

Tournez la page S.V.P.

Permis	PTAC	Véhicule isolé	Remorque	Âge minimum	Équivalence
C1	3,5 à 7,5 t	oui	< 750 kg	18	-
C1E	$> 3{,}5$ t, max 12t	non	> 750 kg	18	-
C	$> 3{,}5$ t	oui	< 750 kg	21	C1
CE	$> 3{,}5$ t	non	> 750 kg	21	C1E

TABLE 1 – Les permis poids lourd pour le transport des marchandises.

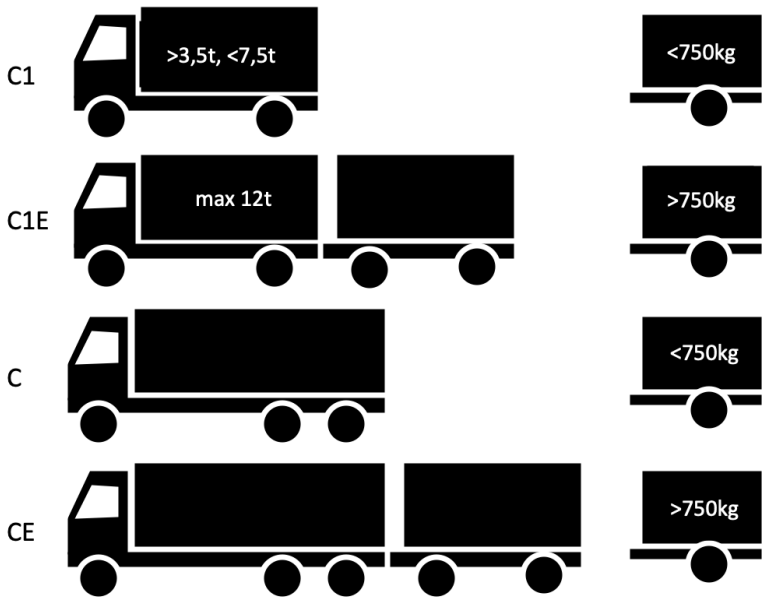


FIGURE 3 – Les permis poids lourd pour le transport de marchandises.

automatique de compatibilité d'affectation entre un véhicule et un conducteur. Pour ceci, il est nécessaire de pouvoir stocker les spécifications des différents permis dans la base et il a été décidé d'ajouter une table `Permis` au schéma relationnel.

Question 27. Modéliser la nouvelle table `Permis` sous une forme similaire au listing 4 en précisant le domaine pour chaque attribut. (Pas de requête en SQL.)

Question 28. Une analyse plus approfondie du schéma relationnel montre que le type d'un véhicule est stocké sous la forme d'une simple chaîne de caractères. Discuter les limites de ce choix de modélisation.

L'entreprise a fait un inventaire de sa flotte de véhicules et il s'est avéré que les véhicules peuvent être catégorisés selon les types suivants :

Type	longueur	largeur	hauteur	PTAC	capacité palettes 80x120
Semi-remorque	16,5 m	2,55 m	4 m	26 t	33
Porteur 19 t	10,70 m	2,55 m	4 m	19 t	20
Porteur 12 t	9,5 m	2,55 m	4 m	12 t	14
20 m^3	7 m	2,5 m	3,10 m	3,5 t	8

Question 29. Proposez une extension du schéma relationnel qui permet la vérification automatique du permis de conduire lors d'une affectation d'un conducteur à un véhicule. Présenter votre proposition sous une forme similaire à la figure 2 et au listing 4. Justifiez vos choix.

Partie V. Utilisation des éléments mobiles

Dans cette partie, nous allons mettre en place une application web qui permettra à un chauffeur de visualiser ses retards dans sa tournée, ainsi que de valider ses livraisons.

Un aperçu de l'application est donné dans la figure 4. Le chauffeur verra la liste des clients à livrer avec l'heure de livraison. Le code couleur suivant permettra de visualiser directement l'état de la livraison :

blanc Le créneau de la livraison est dans le futur.

gris clair Le créneau de la livraison est dépassé d'une heure au plus.

gris foncé Le créneau de livraison est largement dépassé (plus d'une heure).

noir La livraison a été effectuée.

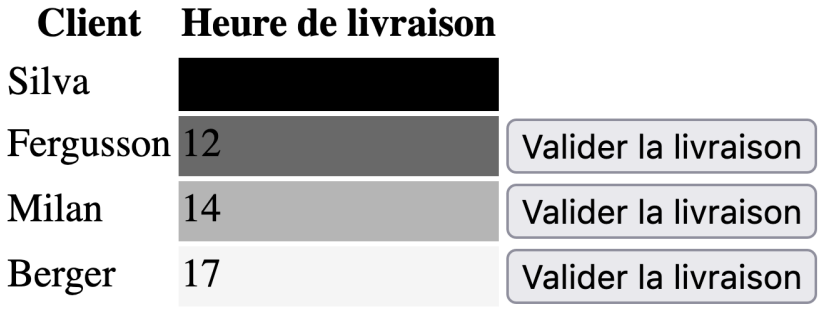


FIGURE 4 – Application web : Visualisation de tournée, heure actuelle : 14h13.

Le bouton **Valider la livraison** permet de changer le statut de la livraison et l'affichage du créneau horaire passe en noir.

Ainsi, on comprend dans l'exemple de la figure 4 que la livraison de Silva a été validée. La livraison pour Fergusson est très en retard tandis que la livraison pour Milan est un peu en retard. La livraison pour Berger peut être effectuée sans retard.

Question 30. En utilisant HTML et CSS uniquement, programmer la partie statique de la page, c'est-à-dire l'affichage du tableau sans la logique des changements de couleur en supposant qu'il n'y a que les quatre clients visibles sur la figure 4 à afficher.

Tournez la page S.V.P.

Listing 5 – Début du code de la page statique.

```
<!DOCTYPE html>
<html>
    <head>
        <style>
                .ontime {
                        background-color: white;
                }
                .hurry {
                        background-color:lightgrey;
                }
                .late {
                        background-color: dimgray;
                }
                .done {
                        background-color: black;
                }
        </style>
     </head>
     <body>
          <!-- TODO -->
     </body>
</html>
```

Question 31. En utilisant JavaScript, programmer l'affichage en couleurs des créneaux de livraison par rapport à l'heure actuelle. La première ligne du code est déjà écrite (voir listing 6).

Listing 6 – Début du code de la coloration automatique des créneaux de livraison.

```
  <script>
  let hours = document.querySelectorAll('.heure')

  const d = new Date()
  let heureActuelle = d.getHours()

  //TODO
  </script>
```

Question 32. En utilisant JavaScript, programmer la validation d'une livraison : dès que le bouton **Valider la livraison** a été actionné, le créneau de livraison s'affiche en noir et le bouton disparaît. La première ligne du code est déjà écrite (voir listing 7).

Listing 7 – Début du code de la fonctionnalité du bouton.

```
  <script>
  let rows = document.querySelectorAll(".row")
  //TODO
  </script>
```

Il arrive aux conducteurs de vouloir changer le créneau de livraison pour un client dans l'application. Dans ce cas, le conducteur saisit le nom du client ainsi que le nouveau créneau dans un formulaire avec deux champs dans l'application. L'application envoie une requête HTTP vers un serveur web qui s'occupe de vérifier si le changement de créneau est possible. Plus précisément, l'application envoie une requête GET vers /**traitement** avec les paramètres **client** et **nHeure** avec les données fournies par le conducteur via le formulaire web. Si on souhaite changer l'heure du client **Fergusson** à 18h et en supposant que l'adresse du serveur est `http://www.appli-tournee.fr`, alors un telle requête serait : `GET //www.appli-tournee.fr/traitement?client=Fergusson&nHeure=18` . Si la réponse du serveur est positive, alors l'affichage du créneau horaire du client doit changer sur le site. Par contre, en cas de réponse négative, une alerte sera donnée au conducteur signalant que le changement n'est pas possible. On rappelle que la fonction JavaScript "alert(message)" permet de lancer une alerte à l'utilisateur.

Question 33. En utilisant des fonctions asynchrones de JavaScript, programmer le script qui permet de changer le créneau horaire d'un client. Le listing 8 donne la première ligne du code.

Listing 8 – Début du code pour mettre à jour un créneau de livraison.

```
<script>
let buttonMAJ = document.getElementById("maj")
//TODO
</script>
```

Rappels

- `element.innerHTML` : la propriété `Element.innerHTML` de `Element` récupère ou définit la syntaxe HTML décrivant les descendants de l'élément.
- `EventTarget.addEventListener()` : la méthode `addEventListener()` d'un `EventTarget` attache une fonction à appeler chaque fois que l'évènement spécifié est envoyé à la cible.
- `document.querySelector` : la méthode `querySelector()` de l'interface `Document` renvoie le premier `Element` dans le document correspondant au sélecteur - ou groupe de sélecteurs - spécifié(s), ou `null` si aucune correspondance n'est trouvée.