Exercice 3 (8 points)

Cet exercice porte principalement sur les bases de données, les graphes et la programmation de base en Python.

Un supermarché utilise une base de données qui contient des informations sur les produits, les fournisseurs, les commandes passées et leurs détails. Le modèle relationnel de cette base est donné par le schéma ci-dessous :

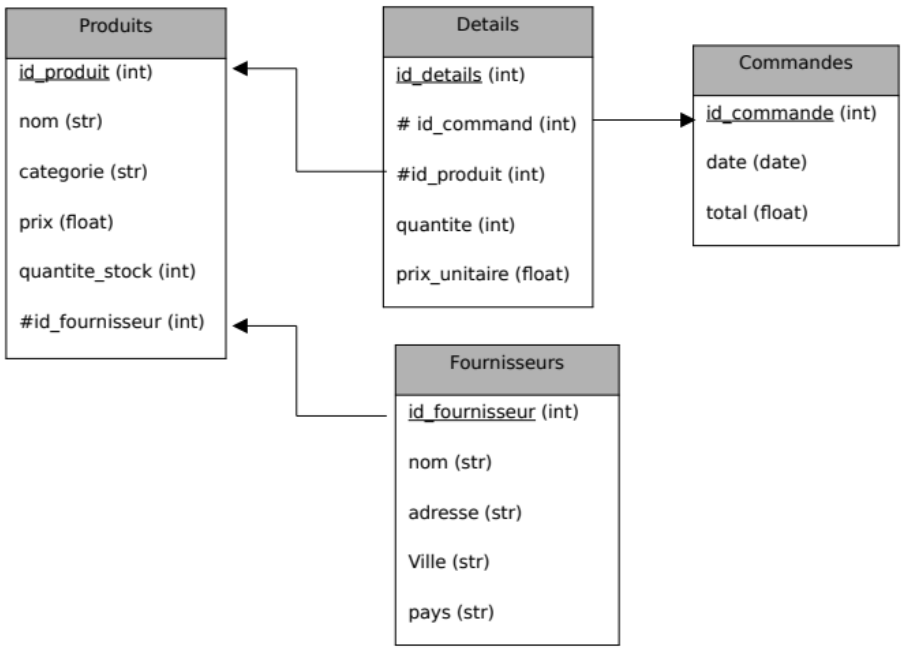

Figure 1. Schéma relationnel de cette base

Dans ce schéma, les clés primaires sont soulignées et les clés étrangères sont précédées du symbole #. Le type de chaque attribut est indiqué entre parenthèses.

On considère l'extrait de la base de données ci-dessous :

Table `Commandes`

id_commande	date_commande	total_commande
1	03/06/2025	176.00
2	08/12/2024	1150.00
3	21/04/2025	155.00

Table `Produits`

id_produit	nom	categorie	prix	quantite_stock	id_fournisseur
1	Yaourts blanc x 4	Alimentaire	2.80	50	2
2	Lait	Alimentaire	1.20	200	2
3	Pain	Alimentaire	1.50	100	4
4	Harry Potter 1	Livre	15.00	20	3
5	Jeu d’échecs	Jeux	40.00	30	3
6	T-shirt taille M	Vêtement	10.00	80	1
7	Jeans taille M	Vêtement	25.00	60	5

Table `Fournisseurs`

id_fournisseur	nom	adresse	ville	pays
1	Moda e stile	Via della Moda, 45	Milano	Italie
2	Laiteries Unies	22 Avenue des Vaches	Lisieux	France
3	Livres en Folie	56 Boulevard des Livres	Toulouse	France
4	Boulangerie du Coin	34 Rue du Pain	Nantes	France
5	Estilo Español	Calle de la Moda, 123	Madrid	Espagne

Table `Details`

id_details	id_commande	id_produit	quantite	prix_unitaire
1	1	1	20	2.80
2	1	2	100	1.20
3	2	6	40	10.00
4	2	7	30	25.00
5	3	5	2	40.00
6	3	4	5	15.00

L'énoncé de cet exercice utilise tout ou une partie des mots clefs du langage SQL suivants : `SELECT, DISTINCT, FROM, WHERE, JOIN … ON, UPDATE … SET, DELETE, INSERT INTO … VALUES.`

Avant de mettre en vente un nouveau produit, il faut le créer dans la base.

1. Écrire une requête SQL permettant d'ajouter le produit *croissant* référencé dans la base sous le numéro 10. Il est vendu au prix unitaire de 0,90 €. Le magasin se fournit, pour ce produit, auprès de *Boulangerie du Coin*.

Le fournisseur *Livres en Folie* a changé d'entrepôt. Il se trouve maintenant au 78 Rue des Jeux à Elbeuf (France).

2. Écrire la requête SQL permettant de mettre à jour la base de données.

3. Décrire le résultat obtenu avec la requête SQL ci-dessous :

```
SELECT nom
FROM Produits
WHERE categorie = 'Alimentaire' ;
```

4. Écrire une requête SQL permettant d'afficher les détails des commandes passées ayant un total de commande supérieur ou égal à 1000 €.

5. Écrire une requête SQL permettant d'afficher le nom des fournisseurs basés en Espagne ou en Italie.

6. Écrire une requête SQL qui permet d'afficher le nom de tous les fournisseurs qui ont vendu des produits alimentaires.

7. Écrire une requête SQL permettant d'afficher le numéro et la date des commandes ainsi que le nom des fournisseurs où des produits de catégories `Vêtement` ont été commandés.

Un fournisseur, dont l'entrepôt est situé à Toulouse, approvisionne différents supermarchés à travers la France, notamment dans les villes de Bordeaux, Calais, Lyon, Marseille, Nantes, Paris et Strasbourg.

On utilise un graphe dont les sommets sont les initiales des villes où se situent les supermarchés et l'entrepôt du fournisseur. Les arêtes sont pondérées avec les distances en kilomètres entre les villes.

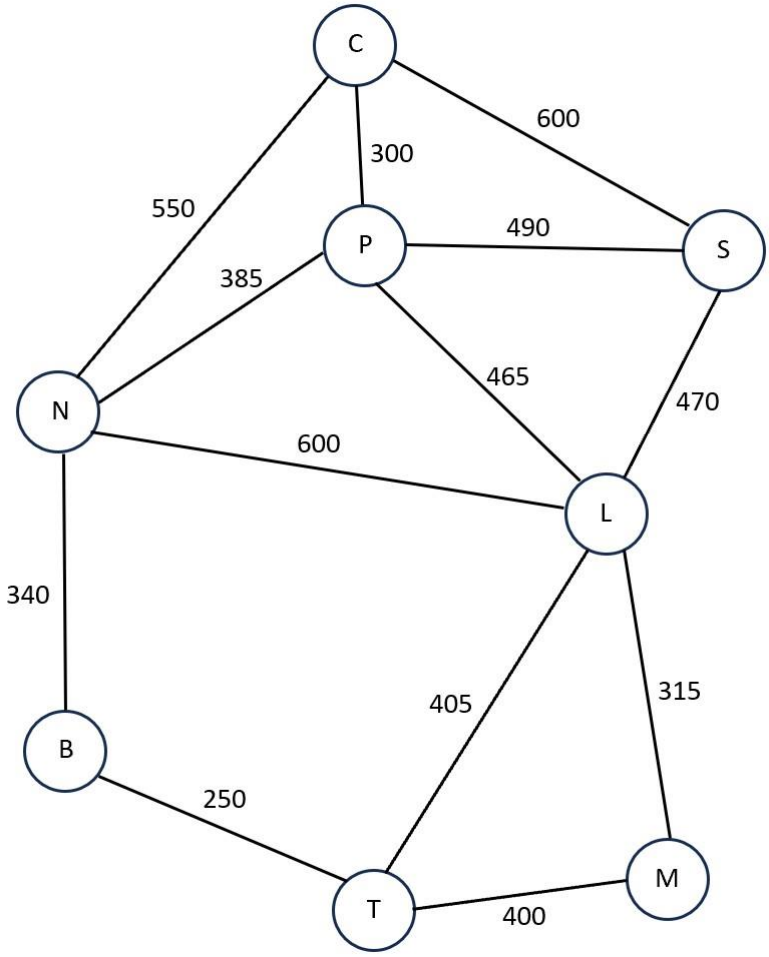

Figure 2. Graphe représentant le réseau routier

8. Déterminer le plus court chemin (en termes de distance) entre Toulouse et Calais.

9. Écrire la liste des sommets dans l'ordre d'un parcours en profondeur à partir de Calais (les sommets sont toujours pris dans l'ordre alphabétique s'il y a un choix à faire).

10. Écrire la liste des sommets dans l'ordre d'un parcours en largeur à partir de Calais (les sommets sont toujours pris dans l'ordre alphabétique s'il y a un choix à faire).

Pour implémenter ce graphe, on utilise le dictionnaire en Python ci-dessous :

```
graphe = {'Paris': {'Strasbourg': 490, 'Lyon': 465,
                    'Nantes': 385, 'Calais': 300},
          'Strasbourg': {'Paris': 490, 'Lyon': 470,
                         'Calais': 600},
          'Lyon': {'Paris': 465, 'Strasbourg': 470,
                   'Toulouse': 405, 'Nantes': 600},
          'Nantes': {'Paris': 385, 'Lyon': 600,
                     'Bordeaux': 340, 'Calais': 550},
          'Calais': {'Paris': 300, 'Strasbourg': 600,
                     'Nantes': 550},
          'Toulouse': {'Lyon': 405, 'Bordeaux': 250,
                       'Marseille': 400},
          'Marseille': {'Toulouse': 400},
          'Bordeaux': {'Nantes': 340, 'Toulouse': 250}
}
```

Dans cette implémentation, il manque la route entre Lyon et Marseille.

11. Écrire les instructions permettant d'ajouter dans le dictionnaire `graphe` la route entre les villes de Lyon et Marseille sachant que la distance les séparant est de 315 km.

12. Écrire une fonction `distance` qui prend en paramètres le dictionnaire `graphe` et deux villes (de type `str`) et qui renvoie la distance entre ces deux villes si elles sont adjacentes, ou `None` sinon.

Pour déterminer le chemin entre deux villes quelconques (s'il en existe un) dans le dictionnaire `graphe`, on utilise la fonction `trouver_chemin(graphe, ville_depart, ville_destination)` qui renvoie la liste des villes parcourues. On suppose que cette fonction est codée en Python.

13. Écrire une fonction `distance_totale` qui prend en paramètre le dictionnaire `graphe` et deux villes (de type `str`) et qui renvoie la distance entre ces deux villes s'il existe un chemin entre elles sinon elle retourne `None`.

On décide désormais de prendre en compte le temps nécessaire pour parcourir les distances entre les villes.

Ainsi les arêtes du graphe sont pondérées à l'aide d'une liste contenant la distance (en km arrondie à l'unité) et la durée (en heure arrondie à deux décimales) nécessaire pour effectuer le trajet entre les deux villes.

Exemple :
```
>>> graphe['Paris']
{'Strasbourg': [490, 6.37], 'Lyon': [465, 5.58], 'Nantes':
[385, 3.47], 'Calais': [300, 3.3]}
```

14. À partir de l'exemple précédent, déterminer la valeur de `graphe['Paris']['Nantes'][1]`.

La fonction `ratio_duree_distance` ci-dessous prend en paramètre le dictionnaire `graphe`. Elle permet de calculer le ratio durée/distance pour toutes les arêtes du graphe et de l'ajouter à la pondération de chaque arête :

```
1 def ratio_duree_distance(graphe):
2     for ville, connexions in ...:
3         for destination, valeurs in ...:
4             distance, duree = ...
5             ratio = ...
6             graphe[ville][destination].append(...)
7     return graphe
```

Grâce à cette fonction, on obtient la mise à jour du dictionnaire graphe :

```
graphe = {'Paris': {'Strasbourg': [490, 6.37, 0.013],
                    'Lyon': [465, 5.58, 0.012],
                    'Nantes': [385, 3.47, 0.009],
                    'Calais': [300, 3.3, 0.011]},
          'Strasbourg': {'Paris': [490, 6.37, 0.013],
                         'Lyon': [470, 4.23, 0.009],
                         'Calais': [600, 9.0, 0.015]},
          'Lyon': {'Paris': [465, 5.58, 0.012],
                   'Strasbourg': [470, 4.23, 0.009],
                   'Toulouse': [405, 4.86, 0.012],
                   'Marseille': [315, 2.84, 0.009],
                   'Nantes': [600, 7.2, 0.012]},
          'Nantes': {'Paris': [385, 3.47, 0.009],
                     'Lyon': [600, 7.2, 0.012],
                     'Bordeaux': [340, 4.08, 0.012],
                     'Calais': [550, 8.25, 0.015]},
          'Calais': {'Paris': [300, 3.3, 0.011],
                     'Strasbourg': [600, 9.0, 0.015],
                     'Nantes': [550, 8.25, 0.015]},
          'Toulouse': {'Lyon': [405, 4.86, 0.012],
                       'Bordeaux': [250, 2.5, 0.010],
                       'Marseille': [400, 6.0, 0.015]},
          'Marseille': {'Lyon': [315, 2.84, 0.009],
                        'Toulouse': [400, 6.0, 0.015]},
          'Bordeaux': {'Nantes': [340, 4.08, 0.012],
                       'Toulouse': [250, 2.5, 0.010]}
}
```

15. Recopier et compléter la fonction `ratio_duree_distance`.

Un élève souhaite utiliser ChatGPT pour trouver un algorithme qui détermine le chemin à privilégier entre deux villes. Il fournit son script Python contenant le dictionnaire `graphe` (et son descriptif) et les fonctions précédentes. Il écrit le prompt suivant :

Écris un algorithme, en langage naturel, pour trouver un chemin entre deux villes en minimisant le ratio durée/distance où à chaque étape, on choisira l'arête avec le ratio le plus faible.

16. Déterminer le nom que l'on donne à un algorithme qui construit une solution étape par étape, comme celui demandé par l'élève.

17. Déterminer le chemin trouvé grâce à cet algorithme entre Toulouse et Calais.