Amérique du Nord – 2025 – sujet2 - Correction

Exercice 3 (8 points)

Partie A – Base de données

1. Clé primaire de la table reservation

L'attribut id_vol ne peut pas être une clé primaire de la table reservation car un même vol peut être réservé par plusieurs passagers. Il apparaît donc plusieurs fois dans la table.

---

2. Clé possible

Une clé primaire possible pour la table reservation est le couple (id_vol, id_passager), qui identifie de manière unique une réservation.

---

3. Rôle d'une clé étrangère

Une clé étrangère permet d'assurer l'intégrité référentielle : elle garantit que la valeur stockée correspond à une clé primaire existante dans une autre table.

---

4. Résultat de la requête SQL

La requête renvoie les identifiants des vols ayant pour aéroport d'arrivée CDG :
AI0015, AI0258, AI0292.

---

5. Villes destination depuis CDG

```
SELECT DISTINCT aeroport.ville
FROM vol
JOIN aeroport ON vol.aeroport_arr = aeroport.id_aeroport
WHERE vol.aeroport_dep = 'CDG';
```

---

6. Mise à jour de la distance totale

```
UPDATE passager
SET d_totale = d_totale + 6
WHERE id_passager = 5;
```

---

7. Correction de l'erreur d'insertion

L'erreur provient de l'utilisation d'un identifiant de vol déjà existant (AI0256 est déjà présent dans la table). Il faut utiliser un nouvel identifiant de vol.

```
INSERT INTO vol
VALUES ('AI1900', 'CDG', 'YUL', 6);
```

Partie B – Graphes pondérés

8. Valeur de graphe_airinfo['T']['P']

La valeur est 10, correspondant à la distance entre Tokyo et Paris.

---

9. Fonction vol_direct

```
def vol_direct(graphe, ville1, ville2):
    return ville2 in graphe[ville1]
```

👉*Commentaire : on teste simplement l'existence de la clé ville2 dans le dictionnaire des voisins de ville1.*

---

10. Fonction liste_villes_proches

```
def liste_villes_proches(graphe, ville, d_max):
    proches = []
    for v in graphe[ville]:
        if graphe[ville][v] <= d_max:
            proches.append(v)
    return proches
```

👉*Commentaire : on parcourt les voisins et on sélectionne ceux dont la distance est inférieure ou égale au seuil.*

---

112. Graphe `Droidevant`

Le graphe comporte deux composantes connexes : W–P–B et T–S.

---

12. Connexité des graphes

La bonne réponse est la proposition A : le graphe de la compagnie AirInfo est connexe.

---

13. Fonction parcours

La fonction est récursive car elle s'appelle elle-même pour explorer les voisins non encore visités d'un sommet.

---

14. Contenu des variables `visitees1` et `visitees2`

```
visitees1 = ['W', 'P', 'T', 'B', 'S']
visitees2 = ['W', 'P', 'B']
```

---

15. Type de parcours

Il s'agit d'un parcours en profondeur (DFS).

---

16. Fonction est_connexe

```
def est_connexe(graphe):
    depart = ville_arbitraire(graphe)
    visitees = []
    parcours(graphe, visitees, depart)
    return len(visitees) == len(graphe)
```

---

17. Affichages produits

Affichage produit par: `mystere (graphe_airinfo, 'W', [], 0, 'B')`

```
['W', 'P', 'T', 'B'] 25
['W', 'P', 'S', 'T', 'B'] 40
```

---

18. Fonction mystere

L'appel `mystere(graphe_airinfo, 'W', [], 0, 'B')` affiche tous les chemins possibles de W vers B ainsi que la distance totale pour chaque chemin.

De manière générale, `mystere(graphe, ville, [], 0, arrivee)` énumère récursivement tous les chemins entre deux villes et calcule leur distance.
