Asie – 2026 – sujet2 - Correction

Exercice 2 (6 points)

Partie A – Base de données

1. Table où id_pers est une clé étrangère

Dans la table **participation** : id_pers y référence la clé primaire id_pers de la table personne.

2. id_partie ne peut pas être la clé primaire de participation

Plusieurs joueurs participent à une même partie : la même valeur de id_partie apparaît donc sur **plusieurs lignes**. Or une clé primaire doit être unique.

👉 *Commentaire : on peut prendre comme clé primaire le couple (id_partie, id_pers) : une personne joue au plus une fois une partie donnée.*

3. Ajout de la personne theorie

```sql
INSERT INTO personne
VALUES (42, 'theorie', '2022-12-14');
```

👉 *Commentaire : la date est une chaîne au format 'AAAA-MM-JJ'. Les valeurs suivent l'ordre id_pers, pseudo_pers, date_pers.*

4. Parties jouées par test

```sql
SELECT participation.id_partie
FROM participation
JOIN personne ON participation.id_pers = personne.id_pers
WHERE personne.pseudo_pers = 'test';
```

👉 *Commentaire : l'attribut id_pers existe dans les deux tables : il faut préciser la table.*

5. Suppression de la personne 8

```sql
DELETE FROM participation WHERE id_pers = 8;
DELETE FROM personne WHERE id_pers = 8;
```

👉 *Commentaire : on supprime d'abord ses participations : sinon elles référenceraient une personne inexistante et la suppression serait refusée.*

Partie B – Ordre alphabétique et graphe

6. Fonction indice (lignes 3 et 4)

```
def indice(lettre, ordre):
    for i in range(len(ordre)):
        if ordre[i] == lettre:
            return i
```

7. Fonction comparer (lignes 7, 9 et 11)

```
def comparer(mot1, mot2, ordre):
    i = 0
    while i < len(mot1) and i < len(mot2):
        i1 = indice(mot1[i], ordre)
        i2 = indice(mot2[i], ordre)
        if i1 < i2:
            return True                         # ligne 7
        elif i1 > i2:
            return False                        # ligne 9
        i += 1
    return len(mot1) < len(mot2)                # ligne 11
```

👉*Commentaire : si la boucle se termine, l'un des mots est un préfixe de l'autre : le plus court est classé en premier. On vérifie les trois exemples de l'énoncé avec l'ordre ouyeai.*

8. Fonction premiere_diff

```
def premiere_diff(mot1, mot2):
    i = 0
    while i < len(mot1) and i < len(mot2) and mot1[i] == mot2[i]:
        i = i + 1
    return i
```

👉*Commentaire : les tests de longueur sont faits avant mot1[i] == mot2[i] : on ne lit jamais en dehors des mots. Si un mot est préfixe de l'autre, la boucle s'arrête à la fin du plus court et renvoie sa longueur.*

9. dico_adj(mots_exemple)

```
{'u': ['a', 'y'], 'y': ['i', 'a'], 'a': ['i'], 'o': ['u'], 'e': ['a']}
```

👉*Commentaire : chaque couple donne un arc, dans l'ordre de la liste : (oou, ooai) donne u → a, (yee, ieo) y → i, (ye, aaaa) y → a, (a, ieo) a → i, (euee, eyy) u → y, (ao, au) o → u et (e, a) e → a.*

10. Type de parcours

C'est un **parcours en profondeur** (récursif) : on explore entièrement un voisin avant de passer au suivant. Un sommet est ajouté à tri seulement quand tous ses successeurs ont été traités (ordre postfixe).

11. Fonction trier

```
def trier(mots):
    adj = dico_adj(mots)
    tri = []
    deja_vus = []
    for voyelle in 'aeiouy':
        if voyelle not in deja_vus:
            deja_vus.append(voyelle)
            parcours(adj, voyelle, deja_vus, tri)
    tri.reverse()
    return tri
```

👉*Commentaire : c’est un **tri topologique** : dans tri, chaque lettre apparaît après toutes celles qui doivent la suivre ; en inversant, on obtient un ordre compatible. Sur mots_exemple, tri vaut ['i', 'a', 'e', 'y', 'u', 'o'] avant inversion, et la fonction renvoie ['o', 'u', 'y', 'e', 'a', 'i'], c’est-à-dire l’ordre ouyeai.*