Métropole - juin 2024 - sujet 2 (corrigé)

Exercice 1 (Bases de données, SQL et protocoles de sécurisation)
Partie A : bases de données

1. On ne peut pas choisir l'attribut `Nom_artiste` comme clé primaire de la relation `CD` car cet attribut ne permet pas d'identifier sans ambiguïté possible une entité/un enregistrement unique de la relation `CD`. En effet un même nom d'artiste (un même artiste ou deux artistes ayant le même nom) peut être associé à plusieurs albums, donc plusieurs entités de la relation `CD`. Pour être éligible en tant que clé primaire l'attribut doit être unique et non vide.
2. La requête renvoie le résultat suivant :

Nightwish
The Rasmus

3. La requête renvoie le résultat suivant :

1986
2001
1986

4. La requête suivante convient :

```
UPDATE CD SET CD.Annee= 2000 WHERE CD.Titre_album = 'Wishmaster'
```

5. La requête suivante convient :

```
SELECT CD.Titre_album FROM CD
JOIN Artiste ON CD.Nom_artiste = Artiste.Nom_artiste
JOIN Rangement ON CD.id_album = Rangement.id_album
WHERE Artiste.Style = 'Metal' AND Rangement.Numero_etagere = 1
```

6. Etant donné qu'il existe des clés étrangères faisant qu'un enregistrement d'une table fait référence à un autre enregistrement d'une autre table, il faut commencer par supprimer l'enregistrement dont l'`id_album = 6` dans la relation `Rangement` à l'aide de la requête :

```
DELETE FROM Rangement WHERE Rangement.id_album = 5
```

Ensuite on supprime l'enregistrement dans la relation `CD` qui fait référence à l'artiste à l'aide de la requête :

```
DELETE FROM CD WHERE CD.id_album = 5
```

Enfin, on supprime l'enregistrement de l'artiste correspondant :

```
DELETE FROM Artiste WHERE Artiste.Nom_artiste = 'The Rasmus'
```

Partie B : sécurisation

7. Un algorithme de chiffrement symétrique assure la confidentialité des échanges entre un émetteur A et un récepteur B sans qu'une « oreille indiscrète » E puisse connaître le contenu du message. En effet, avant émission, le message a été chiffré par A avec une clé secrète. B possède aussi la clé secrète et il pourra ainsi déchiffrer le message à la réception. E ne possède pas la clé secrète. Cet algorithme est rapide et aujourd'hui implémenté avec AES. Son inconvénient est qu'il faut que la clé secrète soit partagée entre l'émetteur et le récepteur.
8. Le chiffrement asymétrique est basée sur un couple de clés : clé publique (largement diffusée) et la clé privée associée (qui elle ne voyage jamais, reste du côté du destinataire/récepteur du message). Pour que A envoie un message M à B, A chiffre le message avec la clé publique de B. Quand B reçoit le message chiffré, il utilise sa clé privée pour le déchiffrer.
9. Pour que le serveur puisse envoyer à Bob la clé C, il faut que le serveur chiffre le message (la clé C) avec la clé publique de Bob. A réception, Bob déchiffrera le message avec sa clé privé et récupérera la clé C.

Exercice 2 (POO, tris, algorithmes gloutons et récursivité)

Partie A : quelques outils

1. Le code suivant convient :

```
def __init__(self, p : int, v :int) -> Marchandise:
    assert v>0, 'le paramètre v doit être strictement positif'
    self.prix = p
    self.volume= v
```

2. Il faut saisir l'instruction : `m1 = Marchandise(20, 7)`
3. La méthode suivante convient :

```
def ratio(self) -> float:
    return self.prix/self.volume
```

4. La méthode suivante convient :

```
def prixListe(tab : list) -> int:
    cumul = 0
    for marchandise in tab:
        cumul += marchandise.prix
    return cumul
```

Partie B : première approche de rangement

5. On a les neuf combinaisons suivantes (toutes les autres dépassent la capacité de 100 litres) :
   - la combinaison vide qui ne rapporte rien ;
   - la combinaison avec uniquement m_1 qui rapporte 40 € ;
   - la combinaison avec uniquement m_2 qui rapporte 210 € ;
   - la combinaison avec uniquement m_3 qui rapporte 190 € ;
   - la combinaison avec uniquement m_4 qui rapporte 50 € ;
   - la combinaison avec m_1 et m_2 qui rapporte 250 € ;
   - la combinaison avec m_1 et m_3 qui rapporte 200 € ;
   - la combinaison avec m_1 et m_4 qui rapporte 90 € ;
   - la combinaison avec m_3 et m_4 qui rapporte 210 €.

   On en déduit que la combinaison qui maximise le prix est $m_1 + m_2$ pour un montant de 250 €.
6. Le qualificatif qui s'applique le mieux à l'algorithme est glouton.
7. Le code suivant convient :

```
def tri(tab: list) -> None:
    n = len(tab)
    for i in range(1, n):
        marchandise = tab[i]
        j = i-1
        while j >= 0 and marchandise.ratio() > tab[j].ratio():
            tab[j+1] = tab[j] # décalage à droite
            j = j - 1 # j avance d'un cran vers la gauche
        tab[j+1] = marchandise
```

8. On a affaire à un tri par insertion et donc à un coût quadratique en $O(n^2)$ avec n la taille du tableau.
9. Le code suivant convient :

```
def charge(tab: list, volume: int) -> list:
    tri(tab)
    chargement = []
    n = len(tab)
    for i in range(n):
        if tab[i] <= volume:
            chargement.append(tab[i])
            volume -= tab[i].volume
    return chargement
```

Partie C : rangement optimisé par récursivité

10. Le code suivant convient :

```
def chargeOptimale(tab: list, v_restant: int, i: int) -> list:
    if i >= len(tab):
        return []
    else:
        if tab[i].volume > v_restant:
            return chargeOptimale(tab, v_restant, i+1)
        else:
            option1 = [tab[i]] + chargeOptimale(tab, v_restant - tab[i], i+1)
            option2 = chargeOptimale(tab, v_restant, i+1)
            if prixListe(option1) > prixListe(option2):
                return option1
            else:
                return option2
```

Exercice 3 (POO, graphes et dictionnaires)

Partie A : analyse des classes `Piste` et `Domaine`

1. Les attributs de la classe `Piste` sont les suivants :
   - `nom` de type chaîne de caractères ;
   - `denivele` de type numérique (entier ou flottant) ;
   - `longueur` de type numérique (entier ou flottant) ;
   - `couleur` de type chaîne de caractères ;
   - `ouverte` de type booléen.
2. La méthode suivante convient :

```
def set_couleur(self):
    if self.denivele >= 100:
        self.couleur = 'noire'
    elif self.denivele >= 70:
        self.couleur = 'rouge'
    elif self.denivele >= 40:
        self.couleur = 'bleue'
    else:
        self.couleur = 'verte'
```

3. L'instruction `lievre_blanc.get_pistes()` renvoie une liste d'objets de type `Piste` (Proposition D).
4. Le programme suivant convient :

```
for piste in lievre_blanc.get_pistes():
    if piste.couleur = 'verte':
        piste.ouverte = False
```

5. La fonction suivante convient :

```
def pistes_de_couleur(lst, couleur):
    select_pistes = []
    for piste in lst:
        if piste.couleur == couleur:
            select_pistes.append(piste.nom)
    return select_pistes
```

6. Le code suivant convient :

```
def semi_marathon(L):
    distance = 0
    liste_pistes = lievre_blanc.get_pistes()
    for nom in L:
        for piste in liste_pistes:
            if piste.get_nom() == nom:
                distance = distance + piste.longueur
    return distance > 21.1
```

Partie B : recherche par force brute

7. Il faut saisir l'instruction `domaine['E']['F']` ou l'instruction `domaine['F']['E']` (le graphe est non orienté).
8. La fonction suivante convient :

```
def voisins(G, s):
    return list(G[s].keys())
```

9. Le code suivant convient :

```
def longueur_chemin(G, chemin):
    precedent = chemin[0]
    longueur = 0
    for i in range(1, len(chemin)):
        longueur = longueur + G[precedent][chemin[i]]
        precedent = chemin[i]
    return longueur
```

10. La fonction `parcours` est une fonctino récursive car elle s'appelle elle-même à la ligne 7.
11. Le code suivant convient :

```
def parcours_dep_arr(G, depart, arrivee):
    liste = parcours(G, depart)
    select_chemins = []
    for chemin in liste:
        if chemin[-1] == arrivee and chemin not in select_chemins:
            select_chemins.append(chemin)
    return select_chemins
```

12. Le code suivant convient :

```
def plus_court(G, depart, arrivee):
    liste_chemins = parcours_dep_arr(G, depart, arrivee)
    chemin_plus_court = liste_chemins[0]
    minimum = longueur_chemin(G, chemin_plus_court)
    for chemin in liste_chemins:
        longueur = longueur_chemin(G, chemin)
        if longueur < minimum:
            minimum = longueur
            chemin_plus_court = chemin
    return chemin_plus_court
```

13. Choisir la distance la plus courte est discutable car le but est d'arriver le plus rapidement sur le lieu de l'intervention en urgence. Or ce n'est pas parce qu'un chemin est le plus court qu'il est le plus rapide, surtout en montagne où interviennent les dénivelés. Par exemple, parcourir 1 km en montée avec un important dénivelé positif prend davantage de temps que parcourir 2 kms en forte descente. Il serait intéressant d'utiliser un ratio entre la longueur et le dénivelé.