Amérique du nord - mars 2021 (corrigé)

Exercice 1 (SQL)

1. (a) Cette requête permet d'obtenir la liste des salles et des marques. On aura donc :

```
012     HP
114     Lenovo
223     Dell
223     Dell
223     Dell
```

   (b) Cette requête permet d'obtenir la liste des salles et des marques lorsque les ordinateurs correspondant sont connectés à la vidéo. On aura donc :

```
012     HP
114     Lenovo
223     Dell
```

2. La requête suivante convient :

```
SELECT *
FROM Ordinateur
WHERE annee >= 2017 ORDER BY annee
```

3. (a) L'attribut `salle` ne peut pas être une clé primaire, car on retrouve dans la table `Ordinateur` plusieurs fois la même salle (par exemple la salle 223).

   (b) On a la relation suivante :

```
Imprimante(nom_imprimante : String, marque_imp : String, modele_imp : String,
           salle : String, #nom_ordi : String)
```

4. (a) La requête suivante convient :

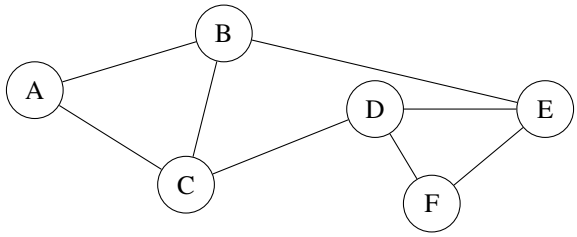
```
INSERT INTO Videoprojecteur(salle, marque_video, modele_video, tni)
VALUES('315', 'NEC', 'ME402X', false)
```

   (b) La requête suivante convient :

```
SELECT Ordinateur.salle, nom_ordi, marque_video
FROM Ordinateur
INNER JOIN Videoprojecteur ON Videoprojecteur.salle = Ordinateur.salle
WHERE video = true AND tni = true
```

Exercice 2 (Routage, processus et SOC)

1. Quelques avantages à utiliser les SOCs :
   - faible consommation d'énergie par rapport à une architecture classique ;
   - miniaturisation par rapport à une architecture classique ;
   - plus faible coût de fabrication qu'une architecture classique.
2. La donnée D2 est utilisée par le SGBD. Dans le même temps, le traitement de texte est en attente de cette même donnée D2. Le SGBD ne peut pas libérer la donnée D2 car il est en attente de la donnée D3 et D4. Cette même donnée D4 est utilisée par le logiciel de CAO qui est lui-même en attente de la donnée D5. Cette donnée D5 est utilisée par le tableur qui est en attente de la donnée D1 alors que cette donnée D1 est utilisée par le tableur. Le tableur est en attente de la donnée D1, donnée D1 utilisée par le traitement de texte. Nous avons donc bien les applications qui s'attendent mutuellement.
   Si un programme A utilise une donnée D et est en attente d'une autre donnée D' alors que dans le même temps un programme B utilise la donnée D' et est en attente de la donnée D, on parle alors d'une situation d'interblocage. C'est cette situation d'interblocage qui nous est décrite dans cette question.
3. On a le chemin suivant : A - B - E - F
4. On a la représentation graphique suivante :



Exercice 3 (Tableaux et programmation)

1. (a) Les deux fonctions suivantes conviennent

```
def total_hors_reduction(tab):
    total = 0
    for pa in tab:
        total = total + pa
    return total
```

```
def total_hors_reduction(tab):
    return sum(tab)
```

(b) La fonction suivante convient :

```
def offre_bienvenue(tab):
    somme = 0
    longueur = len(tab)
    if longueur > 0:
        somme = tab[0]*0.8
    if longueur > 1:
        somme = somme + tab[1]*0.7
    if longueur > 2:
        for i in range(2,longueur):
            somme = somme+tab[i]
    return somme
```

2. Les deux fonctions suivantes conviennent :

```
def prix_solde(tab):
    longueur = len(tab)
    total = total_hors_reduction(tab)
    if longueur == 1:
        return total*0.9
    if longueur == 2:
        return total*0.8
    if longueur == 3:
        return total*0.7
    if longueur == 4:
        return total*0.6
    if longueur >= 5:
        return total*0.5
```

```
def prix_solde(tab):
    return total_hors_reduction(tab)*(1-0.1*len(tab))}
```

3. (a) Les fonctions suivantes conviennent :

```
def minimum(tab):
    mini = tab[0]
    for pa in tab:
        if pa < mini:
            mini = pa
    return mini
```

```
def minimum(tab):
    return min(tab)
```

(b) La fonction suivante convient :

```
def offre_bon_client(tab):
    longueur = len(tab)
    total = total\_hors\_reduction(tab)
    if longueur >= 2:
        total = total - minimum(tab)
    return total
```

4. (a) Le panier `[30.5, 20.0, 35.0, 15.0, 6.0, 5.0, 10.5]` donne un total de $122-20-5=97$ après promotion.
   (b) Le panier `[35, 30.5, 20.0, 15.0, 10.5, 6.0, 5.0]` convient.
   (c) Pour avoir le prix après promotion de déstockage le plus bas possible, il faut trier le tableau dans l'ordre décroissant. On peut donc utiliser un algorithme de tri (tri par sélection, tri par insertion ou tri fusion).

Exercice 4 (Arbres binaires) Pour toutes les questions de l'exercice, on suppose que tous les joueurs d'une même compétition ont un prénom différent.

1. (a) La racine de l'arbre `B` est `"Lea"`. L'ensemble des valeurs de ses feuilles est {`"Marc"`, `"Lea"`, `"Claire"`, `"Theo"`, `"Marie"`, `"Louis"`, `"Anne"`, `"Kevin"`}.

   (b) Comme il est précisé dans la spécification, `arb` n'est pas vide, donc on peut directement renvoyer la racine de l'arbre.

```
def vainqueur(arb):
    return racine(arb)
```

   (c) L'arbre `arb` contenant au moins 2 joueurs, il admet un sous-arbre droit et un sous-arbre gauche non vides.

```
def finale(arb) :
    return [racine(gauche(arb)), racine(droit(arb))]
```

2. (a) Pour compter le nombre d'apparitions du joueur dans l'arbre de compétition, il faut parcourir l'arbre.
   On propose ici une fonction récursive.

```
def occurrences(arb, nom):
    if est_vide(arb) :
        return 0
    elif racine(arb) == nom :
        return 1 + occurences(gauche(arb), nom) + occurences(droit(arb), nom)
    else :
        return occurences(gauche(arb), nom) + occurences(droit(arb), nom)
```

   (b) On utilise la fonction précédente : si le nom du jouer apparaît au moins 2 fois dans l'arbre c'est qu'il a gagné au moins un match.

```
def a_gagne(arb, nom):
    return occurrences(arb, nom) >= 2
```

3. (a) La fonction `occurrences` renvoie le nombre de fois où un joueur donné apparaît dans l'arbre mais ce nombre ne correspond pas toujours au nombre de matchs joués : le gagnant du tournoi apparaît une fois de plus.
   Par exemple `nombre_matchs(B, "Lea")` renvoie en fait `4` et non `3` avec cette implémentation.

   (b) Il suffit de traiter le cas particulier où `nom` est la valeur de la racine de l'arbre passé en argument.

```
def nombre_matchs(arb, nom):
    nb = occurrences(arb, nom)
    if not est_vide(arb) and racine(arb) == nom :
        return nb -1
    return nb
```

4. L'idée ici est de récupérer l'ensemble des feuilles sous la forme d'une liste.

```
def liste_joueurs (arb):
    """arbre_competition -> tableau"""
    if est_vide(arb):
        return []
    elif est_vide(gauche(arb)) and est_vide(droit(arb)):
        return [racine(arb)]
    else:
        return liste_joueurs (gauche(arb)) + liste_joueurs (droit(arb))
```

Exercice 5 (Pile, file et programmation)

1. (a) La file `F` sera vide et la pile `P` sera :

"rouge"
"vert"
"jaune"
"rouge"
"jaune"

(b) La fonction suivante convient :

```
def taille_file(F):
    t = 0
    ft = creer_file_vide()
    while not est_vide(F):
        t = t + 1
        enfiler(ft, defiler(F))
    while not est_vide(ft):
        enfiler(F, defiler(ft))
```

2. La fonction suivante convient :

```
def former_pile(F):
    p = creer_pile_vide()
    pt = creer_pile_vide()
    while not est_vide(F):
        empiler(pt,defiler(F))
    while not est_vide(pt):
        empiler(p,depiler(pt))
    return p
```

3. La fonction suivante convient :

```
def nb_elements(F, ele):
    nb = 0
    ft = creer_file_vide()
    while not est_vide(F):
        x = defiler(F)
        if x == ele:
            nb = nb + 1
        enfiler(ft, x)
    while not est_vide(ft):
        enfiler(F, defiler(ft))
    return nb
```

4. La fonction suivante convient :

```
def verifier_contenu(F, nb_rouge, nb_vert, nb_jaune):
    return nb_elements(F, "rouge") <= nb_rouge and nb_elements(F, "vert") <= nb_vert
           and nb_elements(F, "jaune") <= nb_jaune
```