21NSIJ2AN1

Exercice 1 (4 points)

Cet exercice porte sur les bases de données relationnelles et le langage SQL.

1. (a) `SELECT salle, marque_ordi FROM Ordinateur ;`

012	HP	compaq pro 6300
114	Lenovo	p300
223	Dell	Inspiron Compact
223	Dell	Inspiron Compact
223	Dell	Inspiron Compact

(b) `SELECT nom_ordi, salle FROM Ordinateur WHERE video = true ;`

Gen-24	012
Tech-62	114
Gen-132	223

2. `SELECT * FROM Ordinateur WHERE annee>=2017 ORDER BY annee;` *(On peut aussi écrire >2016)*
3. (a) Parce que il y plusieurs ordinateurs dans une salle et qu'une clé primaire doit être unique.

(b)

Imprimante	
nom_ordi	clé étrangère de la relation Ordinateur
nom_imprimante	
marque_imprimante	
salle	

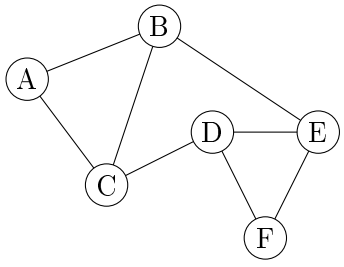
Les clés primaires sont soulignées

4. (a)
```
INSERT INTO Videoprojecteur(salle,marque_video,modele_video,tni)
VALUES (315, « NEC», «ME402X»,false);
```
(b)
```
SELECT Ordinateur.salle, nom_ordi, marque_video FROM Ordinateur JOIN Videoprojecteur
ON Videoprojecteur.salle=Ordinateur.salle WHERE tni= true;
```

Exercice 2 (4 points)

Cet exercice porte sur les notions de routage, de processus et de systèmes sur puces.

1. Pour augmenter encore l'intégration des composants et diminuer la consommation, l'idée est d'intégrer tous les éléments dans une même puce : on obtient un système sur puce ou SoC (system on chip).
   La réduction des distances entre les composants a deux conséquences :
   - la résistance étant proportionnelle à la longueur des circuits et l'énergie consommée par effet Joule augmentant avec cette dernière, la consommation est réduite et il n'est plus nécessaire d'ajouter un système de refroidissement.
   - la circulation des informations gagne en rapidité.
2. CAO est en attente de D5 qui est mobilisé par Tableur, Tableur est en attente de D1 qui est mobilisé par Traitement de Texte, Traitement de Texte est en attente de D2 qui est mobilisé par SGBD, SGBD est en attente de D4 qui est mobilisé par CAO. CAO ne libèrera rien puisqu'il est lui-même en attente.
   On a donc une situation d'**interblocage**.
   Un interblocage (ou deadlock en anglais) est un phénomène qui peut survenir en programmation concurrente. L'interblocage se produit lorsque des processus concurrents s'attendent mutuellement.
   Un processus peut aussi s'attendre lui-même. Les processus bloqués dans cet état le sont définitivement, il s'agit donc d'une situation catastrophique.
3. A - B - E - F
4.



Exercice 3 (4 points)

Cet exercice porte sur les tableaux et sur la programmation de base en Python.

1. (a) On écrit une fonction `total_hors_reduction`.

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

   (b) On complète la fonction `offre_bienvenue`.

```
def offre_bienvenue(tab):
    """ tableau -> float """
    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. On propose ue fonction `prix_solde`.

```
def prix_solde(tab):
    """ tableau -> float """
    somme = total_hors_reduction(tab)
    longueur = len(tab)
    if longueur > 4:
        return somme * 0.5
    elif longueur == 4:
        return somme * 0.6
    elif longueur == 3:
        return somme * 0.7
    elif longueur == 2:
        return somme * 0.8
    elif longueur == 1:
        return somme * 0.9
    elif longueur == 0:
        return somme
```

3. (a) On écrit une fonction `minimum`.

```
def minimum(tab):
    m = tab[0]
    for i in range(1,len(tab)):
        if tab[i] < m:
            m = tab[i]
    return m
```

   (b) On écrit une fonction `offre_bon_client`.

```
def offre_bon_client(tab):
    return total_hors_reduction(tab) - minimum(tab)
```

4. (a) Le panier `tab = [5.0, 6.0, 10.5, 15.0, 20.0, 30.5, 35.0]` donne un prix promotionnel de $6 + 10,5 + 20 + 30,5 + 35 = 102$ euros.

   (b) Le prix le plus bas possible est $35 + 30,5 + 15 + 10,5 + 5 = 91$, obtenu avec e panier `tab=[35.0, 30.5, 20.0, 15.0, 10.5, 6.0, 5.0]`.

(c) L'algorithme pertinent est de trier les prix par prix décroissant, p.ex avec un tri insertion ou sélection ou fusion.

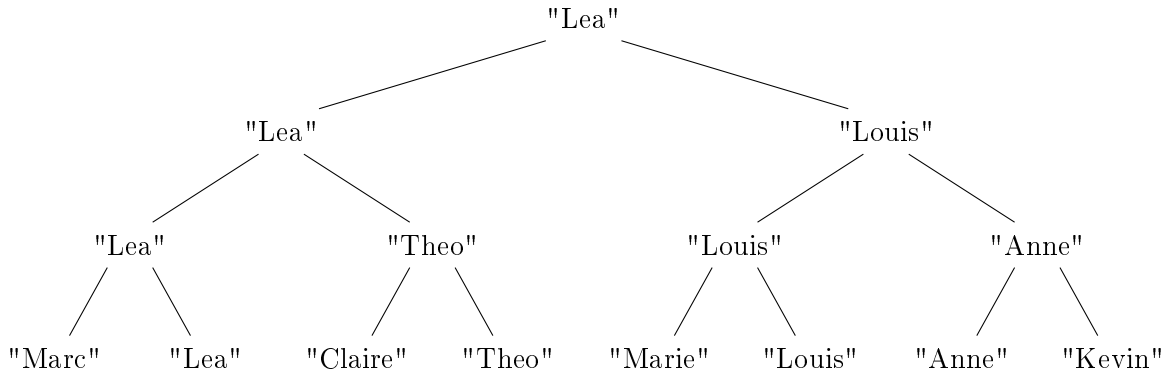
Exercice 4 (4 points)

Cet exercice porte sur les arbres binaires et leurs algorithmes associés.

Les quatre fonctions suivantes pourront être utilisées :

- La fonction `racine` qui prend en paramètre un arbre de compétition `arb` et renvoie la valeur de la racine.
- La fonction `gauche` qui prend en paramètre un arbre de compétition `arb` et renvoie son sous-arbre gauche.
- La fonction `droit` qui prend en argument un arbre de compétition `arb` et renvoie son sous-arbre droit.
- La fonction `est_vide` qui prend en argument un arbre de compétition et renvoie `True` si l'arbre est vide et `False` sinon.

1. (a) On considère l'arbre de compétition `B` suivant :



La racine de cet arbre est "Léa", les valeurs des feuilles sont "Marc", "Lea", "Claire", "Theo", "Marie", "Louis", "Anne" et "Kevin".

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

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

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

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

3. (a) Les instructions proposées renvoient une valeur erronée car pour le vainqueur du tournoi le nombre d'occurrences du nom d'un joueur n'est pas égal au nombre de matchs joués.

(b)
```
def nombre_matchs(arb,nom):
        if nom==racine(arb):
        return occurrences(arb,nom)-1
    else:
        return occurences(arb,nom)
```

4.
```
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 (4 points)

Cet exercice porte sur la notion de pile, de file et sur la programmation de base en Python.

1. (a) Le contenu de la pile P sera celui présenté ci-contre, et la file F sera vide.

P=

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

(b)
```
def taille_file(F):
    n=0
    G=creer_file_vide()
    while not est_vide(F):
        enfiler(G,defiler(F))
        n=n+1
    while not est_vide(G):
        enfiler(F,defiler(G))
    return n
```

2.
```
def former_pile(F):
    P=creer_pile_vide()
    Q=creer_pile_vide()
    while not est_vide(F):
        empiler(P,defiler(F))
    while not est_vide(P):
        empiler(Q,depiler(P))
    return Q
```

3.
```
def nb_elements(F):
    P=former_pile(F)
    n=0
    while not est_vide(P):
        x=depiler(P)
        if x==elt:
            n=n+1
        enfiler(F,x)
    return n
```

4.
```
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
```