Université de Carthage Institut Préparatoire aux Etudes d'Ingénieur de Nabeul Département : Mathématiques	IPEIN المعهد التحضيري للدراسات الهندسية بنابل Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul	Année universitaire : 2019/2020 Filière : SM/SP/ST Niveau d'étude : 2ème année Semestre : 2 Nombre de pages : 7 pages Date : 19/06/2020 Durée : 2h00

Concours blancs : Epreuve d'Informatique ~ Corrigé

Exercice 1 : Les bases de donnes (10 pts)

1. Déterminer les clés primaires et les clés étrangères dans les tables de cette base de données. Justifier votre réponse.

- **Table : Personnes :**
  - Le champ code : clé primaire de la table Personnes
    - Justification : champ à valeurs uniques
    - Le champ code permet d'identifier d'une manière unique un enregistrement de la table Personnes
- **Table : Groupes**
  - Le champ **id** : clé primaire de la table Groupes
    - Justification : champ à valeurs uniques
  - Le champ **codeCr** : clé étrangère, fait référence à la colonne code (clé primaire) de la table Personnes
    - Justification : le champ CodeCr de la table Groupes et le champ code de la table Personnes sont de même type
- **Table : Membres**
  - La clé primaire est composée de deux champs : **code** et **id**
  - Le champ **code** :
    - clé étrangère, fait référence à la colonne code (clé primaire) de la table Personnes
    - Justification : le champ code de la table Personnes et le champ code de la table Membres sont de même type
  - Le champ **id** :
    - clé étrangère, fait référence à la colonne id (clé primaire) de la table Groupes
    - Justification : le champ id de la table Groupes et le champ id de la table Membres sont de même type

Écrire, en langage SQL, les requêtes qui donnent les résultats suivants :

2. Les personnes qui sont nées le mois 6, triées dans l'ordre croissant des âges (du moins âgée au plus âgée).

```sql
SELECT      *
FROM        personnes
WHERE       dateNaiss LIKE '%-06-%'
ORDER BY    dateNaiss DESC
```

3. Les identifiants des groupes, les dates de création des groupes et les noms et prénoms des personnes ayant créé ces groupes.

```sql
SELECT      G.id, G.dateCr, P.nom, P.prenom
FROM        groupes G, personnes P
WHERE       P.code = G.CodeCr
```

4. Écrire la requête de la question **3** en algèbre relationnelle.

$$\pi_{id,dateCr,Personnes.nom,prenom}\left(Groupes \underset{CodeCr = Code}{\rhd\lhd} Personnes\right)$$

5. Les noms et les descriptions des groupes qui contiennent au moins 1000 personnes, triés dans l'ordre décroissant des nombres de personnes.

```sql
SELECT      nom, description
FROM        Groupes G, Membres M
WHERE       G.id = M.id
GROUP BY    G.id
HAVING      COUNT(*) >=1000
ORDER BY    COUNT (*) DESC
```

6. Les codes, les noms et les prénoms de toutes les personnes (créateur et membres) qui appartiennent au groupe ayant le nom : 'Sport'.

```sql
SELECT      P.code, P.nom, P.prenom
FROM        groupes G, personnes P
WHERE       P.code = G.CodeCr AND G.nom='Sport'
UNION
SELECT      P.code, P.nom, P.prenom
FROM        Groupes G, Membres M, personnes P
WHERE       P.code = M.Code AND  G.id = M.id AND G.nom='Sport'
```

7. Supprimer les personnes qui n'appartiennent à aucun groupe.

```sql
DELETE FROM personnes
WHERE       code NOT IN (SELECT  code FROM  membres)
```

8. Modifier dans la table 'Membres', pour que tous les membres du groupe ayant le nom 'Sport' deviennent membres du groupe ayant le nom 'Voyage'.

```sql
UPDATE      membres
SET         id = (SELECT  id FROM  groupes WHERE  nom='Voyage')
WHERE       id = (SELECT  id FROM  groupes WHERE  nom='Sport')
```

Problème : Cryptographie (8 pts)

PARTIE 1 : CHIFFREMENT

1. Ecrire une fonction Python **Taille** permettant de saisir et retourner un entier pair strictement positif et inférieur ou égal à un entier *NMAX* donné passé en paramètres.

```python
def taille(NMAX):
    while True:
        try:
            n=int(input("n = "))
            if 0<=n<=NMAX and n%2==0:
                return n
        except:
            continue
```

2. Ecrire une fonction Python **Code_Car**, qui à partir d'un paramètre **c** de type **str**, retourne l'indice du caractère **c** dans l'**Alphabet** si **c** appartient à l'**Alphabet** et **-1** sinon.

```python
import string
alphabet = list(string.ascii_uppercase)

def code_car(c):
    if c in alphabet :
        return alphabet.index(c)
    else:
        return -1
```

3. Ecrire une fonction Python **saisie_msg** permettant de saisir une chaine de caractères de **n** lettres, en vérifiant l'appartenance à la liste **Alphabet** de chacune des lettres saisies.

```python
def saisie_msg(n):
    while 1:
        msg = input("donner un message de {} lettres : ".format(n))
        if len(msg)!=n:
            continue
        for i in range(n):
            if code_car(msg[i]) == -1:
                break
        else:# si la boucle for n'est pas interrompue avec break
            return msg
```

4. Ecrire une fonction Python **saisie_matc** permettant de saisir la matrice de chiffrement **Mc**.

```
    def saisie_matc():
        mc = np.ones((2,2),dtype=int)
        while 1:
            for i in range(2):
                for j in range(2):
                    while 1:
                        try:
                            c = int(input("Mc[{},{}] = ".format(i,j)))
                            if c >= 0 :
                                mc[i,j] = c
                                break #arrêt de boucle while
                        except:
                            continue
            #vérifier que le det de Mc impair et non mutliple de 13
            if int(np.linalg.det(mc))%2==1 and int(np.linalg.det(mc))%13!=0:
                break #arrêt de boucle while
            else:
                print("Mc est non inversible, saisir Mc à nouveau ...")
        return mc
```

5. Ecrire une fonction Python **chiffrer** permettant de chiffrer dans une chaine de caractères **msg_chiffre**, un message clair donné représenté par une chaine de caractères **msg** de longueur **n**.

```
    def chiffrer(Mc,alphabet, msg):
        msg_chiffre = ""
        n = len(msg)
        i = 0
        while i<n-1:
            p1 = code_car(msg[i])
            p2 = code_car(msg[i+1])
            i += 2

            c1 = int((Mc[0,0] * p1 + Mc[0,1] * p2)%26)
            c2 = int((Mc[1,0] * p1 + Mc[1,1] * p2)%26)

            msg_chiffre += alphabet[c1]
            msg_chiffre += alphabet[c2]

        return msg_chiffre
```

PARTIE 2 : DECHIFFREMENT

6. Ecrire une fonction Python **init_liste**, qui remplit une liste **E** avec des entiers impairs de **1** à **26** et non multiple de de **13**.

```
def init_liste():
    return [i for i in range(27) if i%13!=0]
```

7. Ecrire une fonction Python **Cree_Md** qui crée, à partir de la matrice **Mc** de chiffrement, la matrice **Md** de déchiffrement en utilisant le principe de calcul décrit précédemment.

```
    def cree_Md(Mc):
        a,b,c,d=Mc[0,0],Mc[0,1],Mc[1,0],Mc[1,1]
        Mi = np.array([[d,-b],[-c,a]],dtype=int)
        L = init_liste()
        Md = np.array([])
        for m in L:
            if ((a*d-b*c)*m)%26==1 :
                Md = (m * Mi)%26
                break
        if Md.size == 0 :
            msg_erreur = "Matrice de chiffrement invalide. \n"
            msg_erreur += "Impossible de créer la matrice de déchiffrement!\n"
            msg_erreur += "saisir une autre matrice de chiffrement."
            raise Exception(msg_erreur)
        return Md
```

8. Ecrire une fonction Python **Dechiffrer** permettant de déchiffrer dans une chaine de caractère **msgDc**, un message chiffré donné représenté par une chaine de caractère **msgC** de longueur n.

```
    def dechiffrer(Md,alphabet,msgC):
        msgDc = ""
        n = len(msgC)
        i = 0
        while i<n-1:
            c1 = code_car(msgC[i])
            c2 = code_car(msgC[i+1])
            i += 2

            p1 = int((Md[0,0] * c1 + Md[0,1] * c2)%26)
            p2 = int((Md[1,0] * c1 + Md[1,1] * c2)%26)

            msgDc += alphabet[p1]
            msgDc += alphabet[p2]

        return msgDc
```