Les Bases de Données : SQL

Série d'exercices N° 3 - Corrigé

Exercice 2 :

```
import sqlite3
con=sqlite3.connect("Compte_bancaire.db")
cur=con.cursor()
```

1. Ecrire un script Python permettant de copier les 20 premiers enregistrements de la table Personne dans la table Client.

Question 1 : Première version

```
req="select * from Personne limit 20"
cur.execute(req)

personnes = cur.fetchall()

clients = [(personne[1], personne[2]) for personne in personnes]

req = "insert into client (nomcli, prencli) values (?,?)"
cur.executemany(req,clients)

con.commit()
```

Question 1 : Deuxième version

```
#
req = "select nomp, prenp from personne"
cur.execute(req)
personnes = cur.fetchmany(20)
#
req ="insert into client (nomcli, prencli) values (?,?)"
cur.executemany(req,personnes)
con.commit()
```

Question 1 : Troisième version

```
req = "select * from personne limit 20"
cur.execute(req)
personnes = cur.fetchall()
class Client:
    def __init__(self,nomcli,prencli,numcli=None,adr=None):
        self.numcli = numcli
        self.nomcli = nomcli
        self.prencli= prencli
        self.adr = adr

clients = []
for p in personnes:
    c = Client(nomcli=p[1],prencli=p[2])
    clients.append(c)

liste_clients = [(c.nomcli,c.prencli ) for c in clients]

req = "insert into client (nomcli, prencli) values (?,?)"
cur.executemany(req, liste_clients)

con.commit()
```

2. Ecrire un script Python permettant d'ouvrir un « compte courant » (NomTypC) pour chaque client et d'enregistrer un « versement » (nomTypO) de 500 dinars.

Question 2 : Première version

```
req="select * from client"
cur.execute(req)

clients = []

Class Client:
    def __init__(self, numcli, nomcli, prencli, adrcli):
        self.numcli = numcli
        self.nomcli = nomcli
        self.prencli = prencli
        self.adrcli = adrcli

for tup in cur.fetchall():
    clients.append(Client(tup[0],tup[1], tup[2], tup[3] ))

req="select numTypC from TYPCOMPT where NomTypC LIKE 'compte courant'"
cur.execute(req)
numTypC = cur.fetchone()[0]

req="select numTypO from TYPOPER where nomTypO LIKE 'versement '"
cur.execute(req)
numTypO = cur.fetchone()[0]
```

```
#pour chaque client
for client in clients :
    #ouvrir un « compte courant »
    req = "insert into COMPTE (numcli, numTypC) values (?,?)"
    cur.execute(req,(client.numcli, numTypC))

    #dernier numcpt enregistré ?
    req="select max(numcpt) from COMPTE"
    cur.execute(req)
    dernier_numcpt = cur.fetchone()[0]

    #enregistrer un « versement » de 500 dinars.
    req = """
    insert into OPERATION (numcpt, numcli,numTypO,montant) values (?,?,?,?)
    """
    operation = (dernier_numcpt,client.numcli,numTypO,500)
    cur.execute(req,operation)

con.commit()
```

Question 2 : Deuxième version

```
#chercher tous les clients
cur.execute("select * from client")
liste_tuples_clients = cur.fetchall()

#numéro de type de compte "compte courant"
req = "select numtypc from typcompt
where nomtypc like 'compte courant'
"""
cur.execute(req)
numtypc = cur.fetchone()[0]

#numéro de type de l'opération "versement"
req = "select numtypo from typoper where nomtypo like 'versement'"
```

```
cur.execute(req)
numtypo = cur.fetchone()[0]

#pour chaque client
for client_tup in liste_tuples_clients:
    #créer un compte
    req ="insert into compte (numcli, numtypc) values(?,?)"
    cur.execute(req,(client_tup[0],numtypc))
    #récupérer le dernier numéro de compte créé
    cur.execute("select max(numcpt) from compte")
    dernier_numcpt= cur.fetchone()[0]
    #créer une opération de versement de 500 dt
    req ="""
    insert into operation (numcpt,numcli, numtypo,montant)
    values(?,?,?,500)
    """
    cur.execute(req,(dernier_numcpt,client_tup[0],numtypo))
con.commit()
```

3. Ecrire une fonction Python qui reçoit, comme paramètres, 2 numéros de personnes et retourne vrai s'ils sont frères et faux sinon.

Question 3 : Première version

```
import sqlite3
def sont_freres(nump1, nump2):
    con = sqlite3.connect("compte_bancaire.db")
    cur = con.cursor()
    req = "select pere, mere from personne where nump ={}".format(nump1)
    cur.execute(req)
    parents_pers1 = cur.fetchone()
    req = "select pere, mere from personne where nump ={}".format(nump2)
    cur.execute(req)
    parents_pers2 = cur.fetchone()

    return parents_pers1[0] == parents_pers2[0] or  \
           parents_pers1[1] == parents_pers2[1]
```

Question 3 : Deuxième version

```
import sqlite3
def exeSQL(req):
    con = sqlite3.connect("compte_bancaire.db")
    cur = con.cursor()
    cur.execute(req)
    resultat = cur.fetchall()
    return resultat #le résultat toujours liste de tuples
def freres(nump1,nump2):
    req = "select pere, mere from personne where nump = {}".format(nump1)
    pers1 = exeSQL(req)[0]
    req = "select pere, mere from personne where nump = {}".format(nump2)
    pers2 = exeSQL(req)[0]
    return pers1[0]==pers2[0] or pers1[1]==pers2[1]
print(freres(2,3)) #tester la fonction
```

4. Un client peut avoir plusieurs comptes. Ecrire un script Python qui affiche, pour chaque client, son numéro, son nom, son prénom et la liste des comptes qu'il possède (NumCpt, DateOuv).

Question 4 : Première version

```
#étape1 : sélectionner tous les clients
req = "select * from client"
cur.execute(req)
clients = cur.fetchall()

#étape 2 : pour chaque client ==> afficher : num, nom, prenom
for client in clients :
    print("numéro client:",client[0],"nom:",client[1],"prénom:",client[2])
    #étape 2.1 : selectioner les comptes du client de l'itération courante
    req ="select numcpt,dateouv from compte where numcli = {}".\
          format(client[0])
    cur.execute(req)
    comptes = cur.fetchall()
    #étape 2.2 : afficher : num compte,date ouverture de ses comptes
    for compte in comptes :
        print("numcpt : ",compte[0], "Date ouveture :",compte[1])
    print("-"*40) #afficher une ligne de séparation
con.close()
```

Question 4 : Deuxième version

```
import sqlite3
con = sqlite3.connect("compte_bancaire.db")
cur = con.cursor()

class Client:
    def __init__(self,nomcli,prencli,numcli=None,adr=None):
        self.numcli = numcli
        self.nomcli = nomcli
        self.prencli= prencli
        self.adr = adr
    def __str__(self):
        ch = "Numéro Client : {} \n"
        ch += "Nom : {}, "
        ch += "Prénom : {}\n"
        ch += "-"*30
        return ch.format(self.numcli,self.nomcli,self.prencli)

class Compte :
    def __init__(self,numcpt,numcli,dateouv):
        self.numcpt = numcpt
        self.numcli = numcli
        self.dateouv = dateouv
    def __str__(self):
        ch = "Numéro Compte : {}, Date d'ouverture : {}\n"
        return ch.format(self.numcpt,self.dateouv)

cur.execute("select * from client")
liste_tuples_clients = cur.fetchall()

clients = []
for p in liste_tuples_clients:
    c = Client(nomcli=p[1],prencli=p[2],numcli=p[0])
    clients.append(c)

cur.execute("select numcpt,numcli,dateouv from compte")
```

```
liste_tuples_comptes = cur.fetchall()


comptes = []
for p in liste_tuples_comptes:
    c = Compte(p[0],p[1],p[2])
    comptes.append(c)

for client in clients:
    print(client)
    for compte in comptes:
        if compte.numcli == client.numcli:
            print(compte)
    print("-"*50)
```

Question 4 : troisième version

```
import sqlite3
con = sqlite3.connect("compte_bancaire.db")
cur = con.cursor()

req = """ SELECT numcli, nomcli, prencli  from client"""
cur.execute(req)
clients = cur.fetchall()

d=dict()
for client in clients:
    req = """
    SELECT numcpt, dateouv
    FROM compte
    WHERE numcli = {}
    """
    cur.execute(req.format(client[0]))
    comptes_client = cur.fetchall()
    d[client] = comptes_client

for cle,valeur in d.items():
    print("Numéro : {}, Nom : {}, prenom : {}".format(cle[0],cle[1],cle[2]))
    print("\n"+"-"*10)
    for compte in valeur: #valeur : liste de tuples
        print("Num Cpte: {}, date d'ouverture :{}".format(compte[0],
compte[1]))
    print("\n"+"-"*40)
```