DS T NSI POO ET BDD

EXERCICE 1 LA SANDWICHERIE:

Une sandwicherie effectuant des livraisons à domicile souhaite avoir une base de données pour conserver les informations relatives à ses clients et leurs commandes pour optimiser son service de vente à emporter.

Les sandwichs proposés à la vente sont par exemple:

Nom	Prix
Cheeseburger	3.90
Double cheese	4.90
L'italien	4.90
Foie gras Deluxe	15.00
Kebab	6.00

Lorsqu'un client appelle la sandwicherie, l'opérateur qui lui répond demande alors son nom et le recherche dans la base.

Si le client est nouveau, l'opérateur lui demande son prénom, son adresse et son téléphone.

Voici un exemple d'informations que l'on peut avoir sur les clients :

Nom	Prénom	Adresse	Téléphone
Chanthery	Sébastien	1, avenue de la Victoire , Tours	0782757636
Le Terrier	Stéphane	1, rue Raymond Poulidor, Saint Cyr	0617114479

Ensuite l'opérateur prend commande du client: c'est une liste de sandwichs avec les quantités associées. Une commande contient aussi la date et l'heure de la commande.

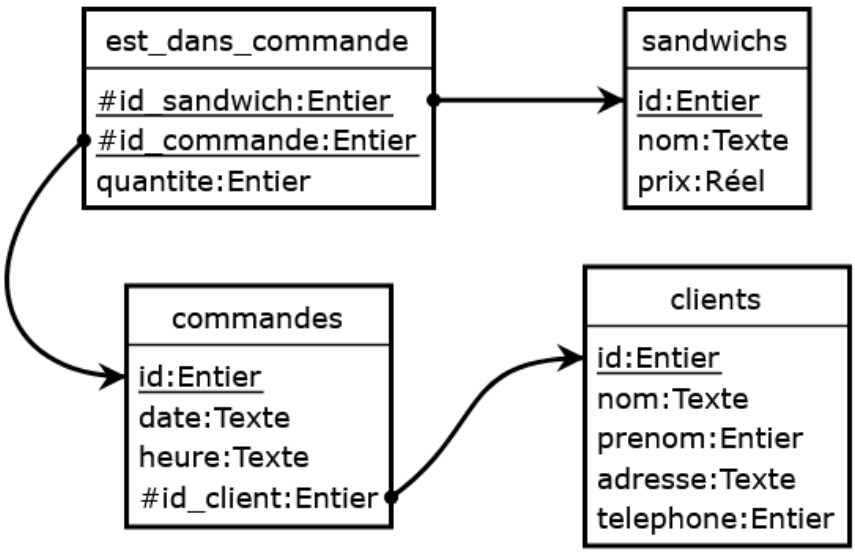
Voici un exemple de commande :

date	22/07/2020	
heure	20 :30	
Client	Chanthery Sébastien	
		Quantité
Commande :	Kebab	1
	Foie gras Deluxe	1
	Double cheese	2

1) Ecrire le modèle physique des données de cette base de données.

Les entités nécessaires sont : sandwichs, client, commandes

L'association nécessaire est : est_dans_commande



2) Ecrire le schéma relationnel ou modèle logique de cette base de données. Vous préciserez les clés primaires, les clés étrangères et leurs références, ainsi que les types des attributs de chaque table.

clients (id:Entier, nom:Texte, prenom:Entier, adresse :Entier, telephone:Entier)

sandwichs (id:Entier, nom:Texte, prix:Réel)

commandes (id:Entier, date:Texte, heure:Texte, #id_client:Entier→clients→id)

est_dans_commande (#id_sandwich:Entier→sandwichs→id, #id_commande:Entier→commandes→id, quantite:Entier)

EXERCICE 2 GESTION DE BIBLIOTHEQUE

Un étudiant en informatique a réalisé ci-dessous le schéma relationnel de la base de données d'une bibliothèque. Vous allez devoir le comprendre et améliorer son travail.

Etudiants(numEtudiant, nomEtudiant, adresseEtudiant)
Livres(idLivre, titreLivre, auteur, editeur, theme, anneeEdition)
Auteurs(idAuteur, nomAuteur, adresseAuteur)
Editeurs(idEditeur, nomEditeur, adresseEditeur)
Themes(idTheme, intituleTheme)
Prets(idEtudiant, idLivre, datePret, dateRetour)

1) Identifiez la clé primaire de la relation **Etudiants**

La clé primaire est numEtudiant

2) Justifiez que idEtudiant ne peut pas être une clé primaire de la relation **Prets**.
   Ce n'est pas possible car si un étudiant emprunte plusieurs livres, il y aura plusieurs enregistrements ayant idEtudiant comme clé primaire ce qui va à l'encontre de la contrainte de relation d'une base de données.

   Même question pour idLivre :
   Pour les mêmes raisons, comme un livre peut être emprunté plusieurs fois, idLivre ne peut être à lui seul la clé primaire

   Même question pour datePret :
   Pour les mêmes raisons, il peut y avoir plusieurs prêts dans une même journée ce qui exclut l'utilisation de datePret seul comme clé primaire

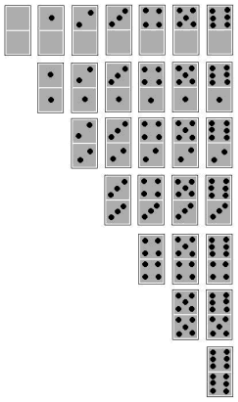
   Quelle clé primaire pouvez-vous proposer pour la relation **Prets** sans rajouter d'attribut ?
   Une combinaison des 3 clés étrangères parait être la solution
   Prets(idEtudiant, idLivre, datePret, dateRetour)

3) Identifiez les deux clés étrangères de la relation **Prets**, en indiquant leur référence.
   idEtudiant est une clé étrangère qui fait référence à l'attribut numEtudiant de la relation Etudiants.
   idLivre est une clé étrangère qui fait référence à l'attribut idLivre de la relation Livres

4) Citez les clés étrangères avec leur référence de la relation **Livres**.
   Les clés étrangères sont :
   - auteur qui fait référence à l'attribut idAuteur de la relation Auteurs
   - editeur qui fait référence à l'attribut idEditeur de la relation Editeurs
   - theme qui fait référence à l'attribut idTheme de la relations Themes
5) Recopiez et complétez le schéma relationnel en indiquant les clés primaires en les soulignant et les clés étrangères en les préfixant par le symbole # et en représentant par des flèches leurs références.

Etudiants(numEtudiant, nomEtudiant, adresseEtudiant)
Livres(idLivre, titreLivre, #auteur→Auteurs→idAuteur, #editeur→Editeurs→idEditeur, #theme→Themes→idTheme, anneeEdition)
Auteurs(idAuteur, nomAuteur, adresseAuteur)
Editeurs(idEditeur, nomEditeur, adresseEditeur)
Themes(idTheme, intituleTheme)
Prets(#idEtudiant→Etudiants→numEtudiant, #idLivre→Livres→idLivre, datePret, dateRetour)

EXERCICE 3



Le code qui vous est donné est issu d'un jeu de dominos. Un domino est une pièce de jeu comportant 2 parties. Chaque partie porte entre 0 et 6 points noirs.
On a créé une classe Domino pour simuler une pièce de ce jeu

PARTIE 1 ANALYSE DE CODE

```
from random import randint
class Domino:
    def __init__(self):
        """
        crée un domino dont les valeurs sont tirés au hasard entre 0 et 6
        """
        self.n1=randint(0,6)
        self.n2=randint(0,6)

    def set_domino(v1,v2):
        """
        permet d'affecter des entiers à un domino
        arguments:
            v1: entier entre 0 et 6
            v2: entier entre 0 et 6
        return:
            None
        exemple:
            domino.set_domino(2,6)
        """
        self.n1=v1
        self.n2=v2

    def __str__(self):
        """
         renvoit une représentation de Domino sous forme de chaîne de
         caractères d'un objet. Elle est appelé lors de l'utilisation de print
        return:
            string
        exemple:
            print(domino)->[5|2]
        """
        return f"[{self.n1}|{self.n2}]"

    def compatible(self,d):
        """
        vérifie si 2 dominos sont compatibles
        argument:
            d:Domino
        return:
            Boolean
        Exemple:
            domino.compatible(domino2)=True
        """
        if self.n1==d.n1 or self.n1==d.n2 or self.n2==d.n1 or self.n2==d.n1:
            return True
        else:
            return False

    def double(self):
        """
        verifie si un domino est un
double
        argument:
            aucun
        return:
            Boolean
        exemple:
            domino.double()=True
        """
        pass

d1=Domino()
d2=Domino()
```

1) Quels sont les attributs de la classe Domino ? Quel est leur type ?

Les aatributs de la classe Domino sont :

- n1 : entier entre 0 et 6
- n2 : entier entre 0 et 6

2) Comment qualifie t'on la fonction `__init__()` ?
   C'est le constructeur
3) Quelles instructions dans le code qui vous a été fourni procède à l'appel de la fonction __init__() ?

```
d1=Domino()
d2=Domino()
```

4) Comment qualifie-t-on alors l'objet obtenu ?
   C'est une instance de la classe
5) Comment s'appelle ce principe de la POO que permet de respecter la fonction set_domino() ?
   L'encapsulation
6) Comment qualifie t'on la fonction compatible() ?
   C'est une méthode de la classe
7) Comment qualifie t'on la fonction `__str__()` ?
   C'est une méthode spéciale

PARTIE 2 SUR MACHINE

8) Compléter la méthode `double()` de la classe Domino. Cette méthode renvoie un booléen : `True` si le domino est un double (2 fois le même nombre) ou `False` sinon.

```
def double(self):
        """
        verifie si un domino est un double ou pas
        argument:
            aucun
        return:
            Boolean
        exemple:
            domino.double()=True
        """
        if self.n1==self.n2:
            return True
        else:
            return False
```

9) Ecrire une classe `Pioche`. Cette classe contient un seul attribut nommé `tas` qui est instancié comme une liste vide à la construction de l'instance de la classe.

Cette classe, outre son constructeur, contiendra 3 autres méthodes :

- La méthode `tirer_tas` : Cette méthode de la classe `Pioche` a pour argument le nombre entier `n` et a pour fonction d'instancier n dominos et de les ranger dans la liste `tas` de l'objet `Pioche`.
- La méthode `affiche` : Cette méthode de la classe `Pioche` n'a besoin d'aucun argument particulier mais affiche à l'aide de la fonction `print()` l'ensemble des dominos contenus dans le `tas` de la `Pioche`.
- La méthode `nombre_de_doubles` : Cette méthode de la classe `Pioche` compte le nombre de dominos de type double contenus dans le `tas` de la `Pïoche`.

On pourra utiliser cette classe de la façon suivante :

```
p=Pioche()
p.tirer_tas(5)
p.affiche()
print("Nombre de doubles:",p.nombre_de_doubles())
```

Et on obtiendra :

[6|4]
[6|2]
[5|3]
[6|6]
[4|3]
Nombre de doubles : 1

```
class Pioche():
    def __init__(self):
        self.tas=[]

    def tirer_tas(self,n):
        for i in range(n):
            d=Domino()
            self.tas.append(d)

    def affiche(self):
        for d in self.tas:
            print(d)

    def nombre_de_doubles(self):
        n=0
        for d in self.tas:
            if d.double()==True:
                n=n+1
        return n
```