Bac Blanc NSI - Corrigé

EXERCICE 1

1. **a)** Une clé primaire d'une relation est un attribut (ou plusieurs attributs) dont la valeur permet d'identifier de manière unique un enregistrment (p-uplet) de la relation.

b) Une clé étrangère est un attribut qui permet d'établir un lien entre deux relations.

c) Un abonné ne peut pas réserver plusieurs fois la même séance, car le couple `(idAbonné, idSéance)` est une clé primaire pour la relation **Réservation**. Il est donc impossible d'avoir deux fois la même séance pour le même abonné.

2. **a)** On obtient :

idAbonné	idSéance	nbPlaces_plein	nbPlaces_réduit
13	737	3	2

b) La requête précédente s'écrit en SQL :

```
INSERT INTO Reservation
VALUES (13, 737, 3, 2);
```

3. Cette requête permet de déterminer le nombre de séances proposées les 10 et 11 janvier 2023.

4. **a)**
```
SELECT nom, prenom FROM Abonne;
```

b)
```
SELECT titre, realisateur
FROM Film
WHERE duree < 120;
```

c)
```
SELECT Film.titre, Film.duree FROM Film
JOIN Seance ON Film.idFilm = Seance.idFilm
WHERE Seance.date = '2023-01-14' AND Seance.heure = '21:00';
```

5. **a)**
```
UPDATE Film
SET duree = 192
WHERE titre = 'Avatar 2';
```

b) `idSéance` est une clé étrangère pour la relation **Réservation**. La suppression d'une séance risque donc de provoquer des problèmes dans la relation **Réservation** (avec un `Réservation.idSéance` ne correspondant à aucun `Séance.idRéservation`). Cela pourrait enfreindre la contrainte de référence.

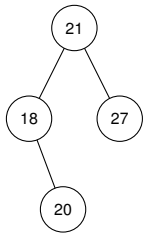
c)
```
DELETE FROM Seance
WHERE idSeance = 135;
```

EXERCICE 2

1. **a)** La taille de cet arbre est 8.

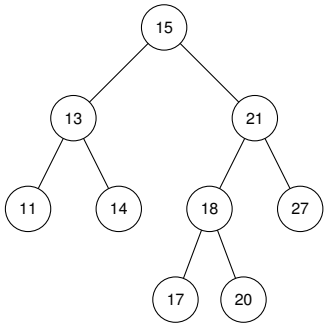
b) La hauteur de cet arbre est 4.

c) On obtient l'arbre suivant :



d) C'est un arbre binaire de recherche car pour tous les nœuds de l'arbre, les valeurs contenues dans le sous-arbre gauche sont inférieures à sa valeur, et les valeurs contenues dans le sous-arbre droit lui sont supérieures.

e) On obtient l'arbre suivant :



2. **a)** C'est la proposition : **(C)** `abr = Noeud(Noeud(None, 13, None), 15, Noeud(None, 21, None))`

b) On insère la valeur à gauche :

```
def inserer(v, abr):
    if abr is None:
        return Noeud(None, v, None)
    if v > abr.valeur:
        return Noeud(abr.gauche, abr.valeur, inserer(v, abr.droite))
    elif v < abr.valeur:
        return Noeud(inserer(v, arbr.gauche), abr.valeur, abr.droite)
    else:
        return abr
```

3. **a)** Quelle que soit la valeur cherchée, la fonction `nb_sup` fait un appel récursif sur les deux fils du nœud. On va donc faire un appel sur chaque nœud, y compris vides, de l'arbre. On aura donc un appel pour chaque nœud non vide, soit 8, plus 9 sur les nœuds vides. Donc 17 appels en tout.

b) Puisque l'arbre passé en paramètre est un arbre binaire de recherche, dans le cas où la valeur cherchée est supérieure à `abr.valeur` , il est inutile de cercher dans le sous-arbre gauche.

On peut donc écrire :

```
def nb_sup(v, abr):
    if abr is None:
        return 0
    elif abr.valeur >= v:
        return 1 + nb_sup(v, abr.gauche) + nb_sup(v, abr.droite)
    else:
        return nb_sup(v, abr.droite)
```

EXERCICE 3

Cet exercice porte sur les structures de données et la programmation objet en langage python.

1. On écrit :

```
panier_1.enfile((31002, "cafe noir", 1.50, 50525))
```

2. Il faut défiler le panier temporaire pour enfiler dans le panier. On obtient

```
def remplir(self, panier_temp):
    while not panier_temp.est_vide():
        article = panier_temp.defile()
        self.enfile(article)
```

3. L'idée est de calculer le prix total en additionnant les prix de chaque article du panier en le vidant et en stockant les articles dans un panier temporaire. Puis on remplira à nouveau le panier comme le suggère la méthode `remplir` précédente.

```
    def prix_total(self):
        prix = 0
        panier_temp = Panier()
        while not self.est_vide():
            article = self.defile()
            prix += article[2]
            panier_temp.enfile(article)
        self.remplir(panier_temp)
        return prix
```

4. On peut supposer que la file est dans l'ordre croissant des horaires de scan.

```
def duree_passage_en_caisse(self):
    if self.est_vide():
        return None
    article = self.defile()
    premier_horaire_scan = article[3]
    while not self.est_vide():
        article = self.defile()
    dernier_horaire_scan = article[3]
    return dernier_horaire_scan - premier_horaire_scan
```