EXERCICE 1 (10 points)

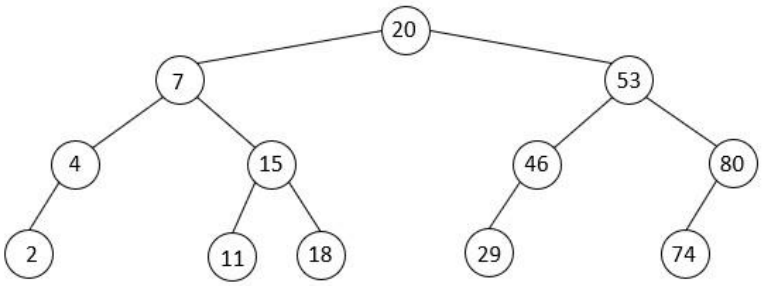
Cet exercice traite des arbres et de l'algorithmique.

Dans cet exercice, la taille d'un arbre est égale au nombre de ses nœuds et on convient que la hauteur d'un arbre ne contenant qu'un nœud vaut 1.

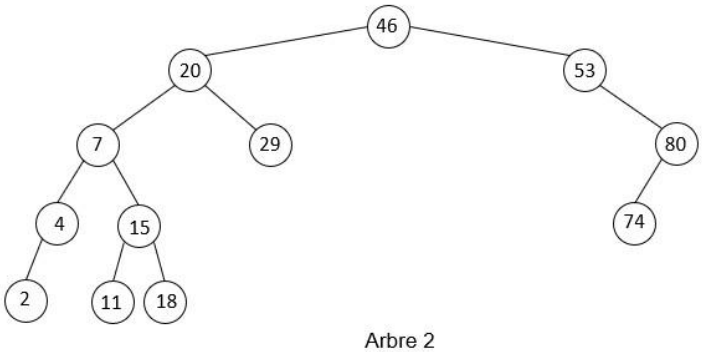
On utilisera la définition suivante : un arbre binaire de recherche est un arbre binaire, dans lequel

- on peut comparer les valeurs des nœuds : ce sont par exemple des nombres entiers, ou des lettres de l'alphabet ;
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre gauche de x, alors il faut que `y.valeur < x.valeur`
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre droit de `x`, alors il faut que `y.valeur` $\geq$ `x.valeur`

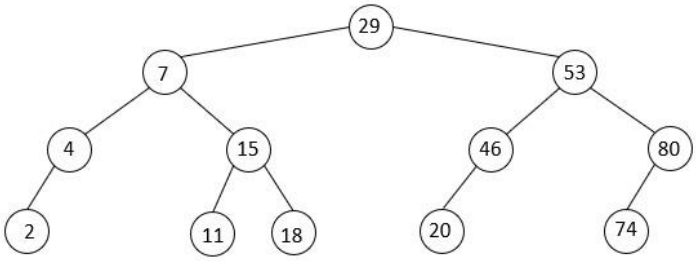
1. Parmi les trois arbres dessinés ci-dessous, recopier sur la copie le numéro correspondant à celui qui n'est pas un arbre binaire de recherche. Justifier



Arbre 1



Arbre 2



Arbre 3

L'arbre 3 est incorrect car le nœud de valeur 20 est supérieur à la racine 29 : il devrait être dans la partie du sous-arbre gauche de la racine 29.

Une classe `ABR`, qui implémente une structure d'arbre binaire de recherche, possède l'interface suivante :

Numéro de lignes	Classe `ABR`
1	`class ABR :`
2	`    def __init__(self, valeur, sa_gauche, sa_droit):`
3	`        self.valeur = valeur #valeur de la racine`
4	`        self.sa_gauche = sa_gauche #sous-arbre gauche`
5	`        self.sa_droit = sa_droit #sous-arbre droit`
6	`    def inserer_noeud(self, valeur):`
7	`        """Renvoie un nouvel ABR avec le nœud de valeur 'valeur'`
8	`        inséré comme nouvelle feuille à sa position correcte"""`
9	`        # code non étudié dans cet exercice`
…	

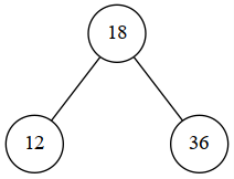
On prendra la valeur `None` pour représenter un sous-arbre vide.

2. La construction d'un ABR se fait en insérant progressivement les valeurs à partir de la racine : la méthode `inserer_noeud` (dont le code n'est pas étudié dans cet exercice) place ainsi un nœud à sa "bonne place" comme feuille dans la structure, sans modifier le reste de la structure. On admet que la position de cette feuille est unique.

   a. En utilisant les méthodes de la classe ABR :
      - écrire l'instruction Python qui permet d'instancier un objet `a0`, de type ABR, ayant un seul nœud (la racine) de valeur 18.

a0 = ABR(18,None,None)

      - écrire une séquence d'instructions qui permet ensuite d'insérer dans l'objet `a0` les deux feuilles de l'arbre de valeurs 12 et 36.



ABR `a0`

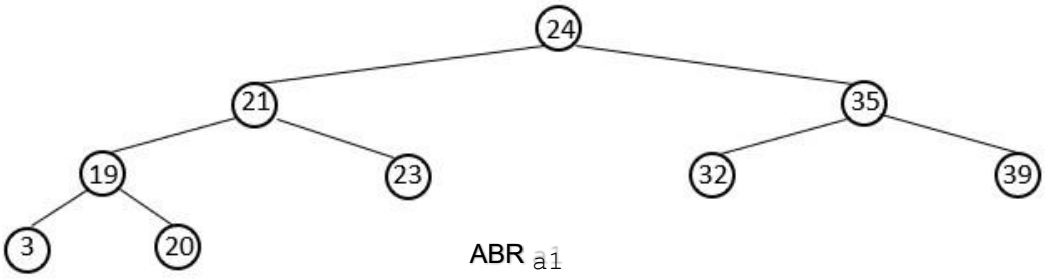
a0.inserer_noeud(12)

a0.inserer_noeud(36)

Selon l'ordre dans lequel les valeurs sont insérées, on construit des ABR ayant des structures différentes.

Voilà par exemple ci-dessous un ABR (nommé `a1`) obtenu en créant une instance de type ABR ayant un seul nœud (la racine) de valeur 24 puis en insérant successivement les valeurs dans l'ordre suivant :

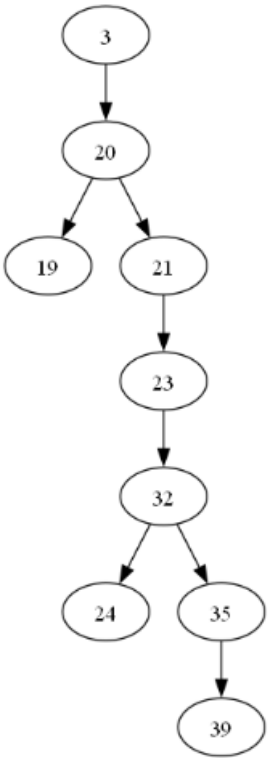
21 ; 35 ; 19 ; 23 ; 32 ; 39 ; 3 ; 20



ABR `a1`

b. Dessiner sur la copie l'ABR (nommé `a2`) que l'on obtiendrait en créant une instance de type ABR ayant un seul nœud (la racine) de valeur 3 puis en insérant successivement les valeurs dans l'ordre suivant :

20 ; 19 ; 21 ; 23 ; 32 ; 24 ; 35 ; 39



c. Donner la hauteur des ABR `a1` et `a2`.

Hauteur de a1 : 4

Hauteur de a2 : 7

d. On complète la classe ABR avec une méthode `calculer_hauteur` qui renvoie la hauteur de l'arbre.

Recopier sur la copie les lignes 10 et 13 en les complétant par des commentaires et la ligne 14 en la complétant par une instruction dans le code ci-après de cette méthode.

On pourra utiliser la fonction Python `max` qui prend en paramètres deux nombres et renvoie le maximum de ces deux nombres.

Numéro de lignes	Méthode `calculer_hauteur`
1	`def calculer_hauteur(self):`
2	`    """ Renvoie la hauteur de l'arbre"""`
3	`    if self.sa_droit is None and self.sa_gauche is None:`
4	`    #l'arbre est réduit à une feuille`
5	`        return 1`
6	`    elif self.sa_droit is None`
7	`    #arbre avec une racine et seulement un sous-arbre gauche`
8	`        return 1 + self.sa_gauche.calculer_hauteur()`
9	`    elif self.sa_gauche is None:`
10	`    # à compléter`
11	`         return 1 + self.sa_droit.calculer_hauteur()`
12	`    else:`
13	`    # à compléter`
14	`         return à compléter`

```python
def calculer_hauteur(self):
    """ renvoie la hauteur de l'arbre """
    if self.sa_droit is None and self.sa_gauche is None:
        #l'arbre est réduit à une feuille
        return 1
    elif self.sa_droit is None:
        #arbre avec une racine et seulement un sous-arbre gauche
        return 1 + self.sa_gauche.calculer_hauteur()
    elif self.sa_gauche is None:
        #arbre avec une racine et seulement un sous-arbre droit
        return 1 + self.sa_droit.calculer_hauteur()
    else:
        #arbre avec une racine, un sous-arbre gauche et un sous-arbre droit
        return 1 + max(self.sa_gauche.calculer_hauteur(),self.sa_droit.calculer_hauteur())
```

3. La différence de hauteur entre l'ABR `a1` et l'ABR `a2` aura des conséquences lors de la recherche d'une valeur dans l'ABR.
   a. Recopier et compléter sur la copie les lignes 6, 8, 11 et 13 du code ci-dessous de la méthode `rechercher_valeur`, qui permet de tester la présence ou l'absence d'une valeur donnée dans l'ABR :

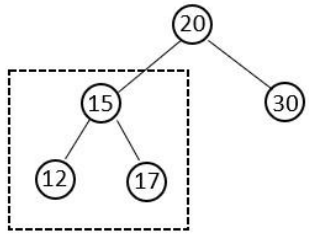
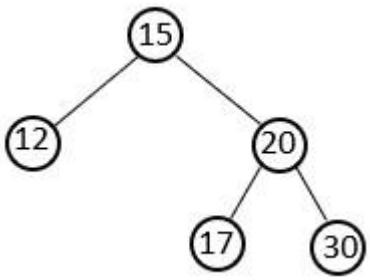
Numéro de lignes	Méthode `rechercher_valeur`
1	`def rechercher_valeur(self, v):`
2	`    """`
3	`    Renvoie True si la valeur v est trouvée dans l'ABR,`
4	`    False sinon`
5	`    """`
6	`    if à compléter`
7	`        return True`
8	`    elif à compléter and self.sa_gauche is not None:`
9	`        return self.sa_gauche.rechercher_valeur(v)`
10	`    elif v > self.valeur and self.sa_droit is not None:`
11	`        return à compléter`
12	`    else:`
13	`        return à compléter`

```
def rechercher_valeur(self,v):
    if self.valeur == v:
        return True
    elif v < self.valeur and self.sa_gauche is not None:
        return self.sa_gauche.rechercher_valeur(v)
    elif v > self.valeur and self.sa_droit is not None:
        return self.sa_droit.rechercher_valeur(v)
    else:
        return False
```

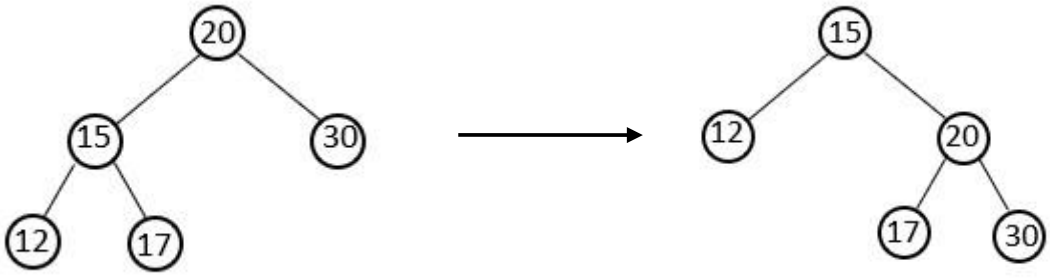
b. On admet que le nombre de fois où la méthode `rechercher_valeur` est appelée pour rechercher la valeur 39 dans l'ABR `a2` est 7.

Donner le nombre de fois où la méthode `rechercher_valeur` est appelée pour rechercher la valeur 20 dans l'ABR `a1`. : 4

4. Il existe des algorithmes pour modifier la structure d'un ABR, afin par exemple de diminuer la hauteur d'un ABR ; on s'intéresse aux algorithmes appelés *rotation*, consistant à faire "pivoter" une partie de l'arbre autour d'un de ses nœuds.

On appelle *pivot* le sousarbre gauche de la racine de l'arbre	 *pivot*
Le sous-arbre droit du pivot devient le sous-arbre gauche de la racine	
La racine ainsi modifiée devient le sous-arbre droit du pivot et la racine du pivot devient la nouvelle racine de l'ABR	

L'exemple ci-dessous permet d'expliquer l'algorithme pour réaliser une rotation droite d'un ABR autour de sa racine :



On admet que ces transformations conservent la propriété d'ABR de l'arbre.

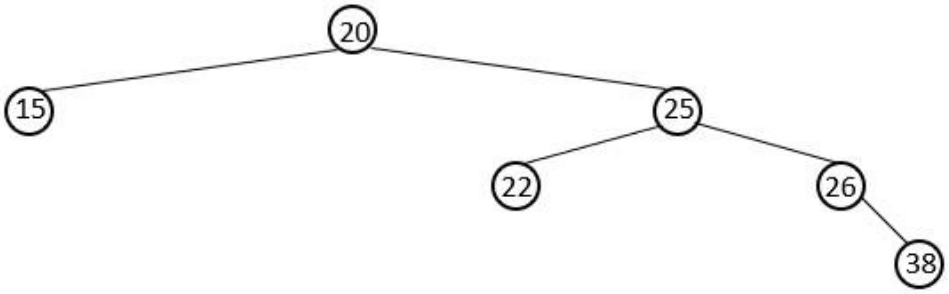
La méthode `rotation_droite` ci-après renvoie une nouvelle instance de type ABR, correspondant à une rotation droite de l'objet de type ABR à partir duquel elle est appelée :

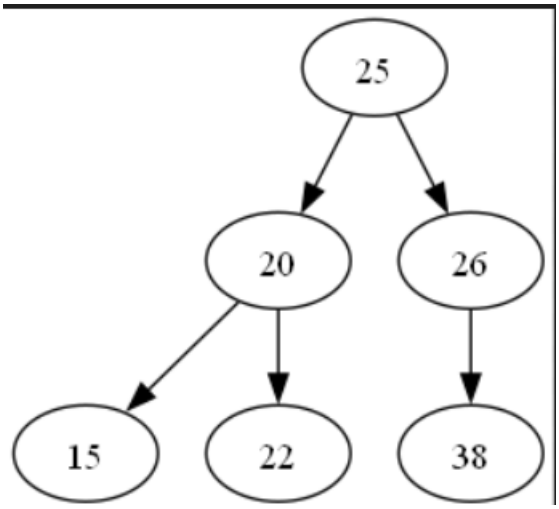
Numéro de lignes	Méthode `rotation_droite`
1	`def rotation_droite(self):`
2	`    """ Renvoie une instance d'un ABR apres une rotation droite`
3	`        On suppose qu'il existe un sous-arbre gauche"""`
4	`    pivot = self.sa_gauche`
5	`    self.sa_gauche = pivot.sa_droit`
6	`    pivot.sa_droit = self`
7	`    return ABR(pivot.valeur,pivot.sa_gauche,pivot.sa_droit)`

Pour réaliser une rotation gauche, on suivra alors l'algorithme suivant :

- on appelle *pivot* le sous-arbre droit de la racine de l'arbre,
- le sous-arbre gauche du pivot devient le sous-arbre droit de la racine,
- la racine ainsi modifiée devient le sous-arbre gauche du pivot et la racine du pivot devient la nouvelle racine de l'ABR

a. En suivant les différentes étapes de cet algorithme, dessiner l'arbre obtenu après une rotation gauche de l'ABR suivant :





b. Écrire le code d'une méthode Python `rotation_gauche` qui réalise la rotation gauche d'un ABR autour de sa racine.

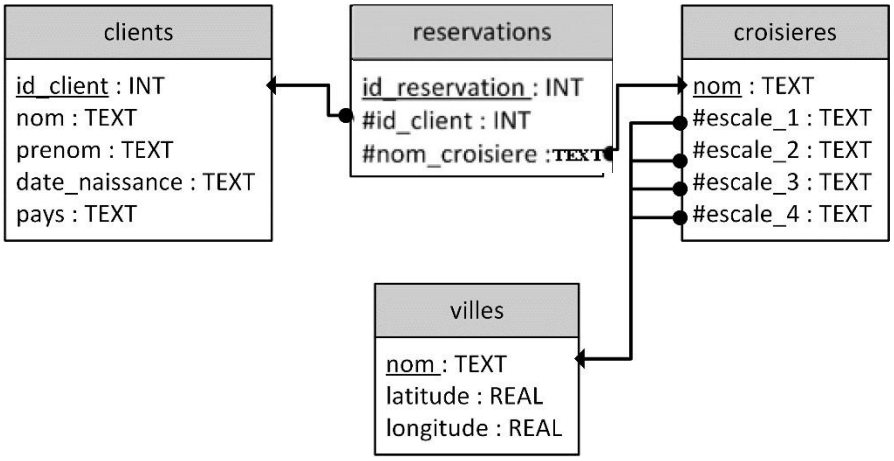
```
def rotation_gauche(self):
    pivot = self.sa_droit
    self.sa_droit = pivot.sa_gauche
    pivot.sa_gauche = self
    return ABR(pivot.valeur,pivot.sa_gauche,pivot.sa_droit)
```

EXERCICE 2 (10 points)

Cet exercice traite de protocoles de routage et de base de données relationnelle.

Une agence de voyage propose des croisières en bateau. Chaque croisière a un nom unique et passe par quatre escales correspondant à des villes qui ont elles aussi des noms différents.

Pour gérer les réservations de ses clients, l'agence utilise une base de données. Voici la description des trois relations de cette base dont les clés primaires ont été soulignées et les clés étrangères indiquées par un # :



Remarque : l'énoncé de cet exercice utilise tout ou une partie des mots suivants du langage SQL : `SELECT`, `FROM`, `WHERE`, `JOIN ON`, `INSERT INTO`, `VALUES`, `UPDATE SET`, `OR`, `AND`.

Partie A

L'agence de voyage possède deux bureaux distincts.

Elle passe par un prestataire de service qui héberge sa base de données et utilise un système de gestion de base de données relationnelle.

Vous trouverez ci-après un schéma du réseau entre les deux bureaux de l'agence de voyage et le prestataire.

On peut y voir les différents routeurs (nommés de A à I) ainsi que le coût des liaisons entre eux.

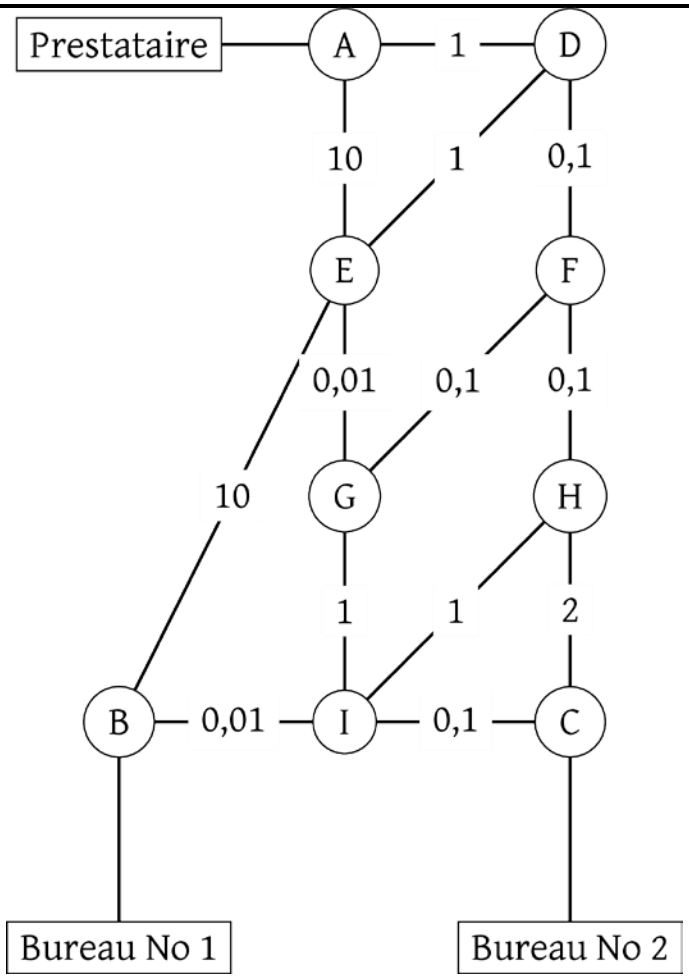


Figure 1. Topologie du réseau

1. Donner deux services rendus par un système de gestion de bases de données relationnelles.

   un SGBD permet de gérer la redondance des données (gestion des pannes) et la sécurisation des accès.

Le protocole RIP (Routing Information Protocol) est un protocole de routage qui minimise le nombre de routeurs par lesquels les paquets transitent.

Le protocole OSPF (Open Shortest Path First) est un protocole de routage qui minimise le coût du transit des paquets.

2. Donner la route suivie par une requête issue du bureau numéro 1 jusqu'au prestataire si on utilise le protocole RIP.

   route : B -> E -> A

3. Donner les deux routes que pourrait suivre une requête issue du bureau numéro 2 jusqu'au prestataire si on utilise le protocole OSPF. Donner le coût de chaque route.

routes :

C -> I -> G -> F -> D -> A ou C -> I -> H -> F -> D -> A avec un coût de 2,3

Partie B

4. Expliquer pourquoi l'attribut `id_client` a été choisi comme clé primaire dans la relation `clients`.

   Parce que l'attribut `id_client` est unique pour chaque entrée de la table `clients`.

5. Définir ce qu'est une clé étrangère. Donner la ou les clés étrangères de chaque relation qui en a en précisant la clé primaire qu'elles référencent.

   Une clé étrangère est un attribut qui permet d'effectuer la jointure entre 2 tables. La clé étrangère d'une table A pointe vers la clé primaire d'une table B ce qui permet d'établir une jointure entre A et B.

L'agence a obtenu l'autorisation de faire escale dans quatre nouvelles villes : *Puerto saibo*, *Puerto kifecho*, *Puerto kifebo* et *Puerto repo*. Elle souhaite créer une nouvelle croisière qui passera par ces quatre villes. Un stagiaire de l'agence demande de l'aide à une *Intelligence Artificielle* (IA) :

Pour ajouter la nouvelle croisière nommée 'Croisière Puerto' avec ses escales correspondantes, vous pouvez utiliser la requête suivante :

sql Copy code

```
INSERT INTO croisieres (nom, escale_1, escale_2, escale_3,
escale_4)
VALUES ('Croisière Puerto',
'Puerto sebo',
'Puerto kifecho',
'Puerto kifebo',
'Puerto repo');
```

Figure 2 - Réponse de l'IA

Il tape alors la requête proposée mais obtient le message d'erreur suivant du SGBD (*Systèmes de Gestion de Bases de Données*) : *FOREIGN KEY constraint failed*.

6. Expliquer l'erreur commise et proposer une solution.

Les attributs escale_1, escale_2… de la table croisieres sont des clés étrangères qui doivent pointer vers des entrées de la table villes. Les villes 'Puerto sebo', 'Puerto kifecho', … n'existent pas dans la table villes, d'où l'erreur. Il est donc

nécessaire de compléter la table villes (avec les villes 'Puerto sebo', 'Puerto kifecho', …) avant de renseigner la table croisieres.

Partie C

7. Jean Barc, un allemand né le 29 juin 1972, demande un geste commercial en raison de sa fidélité à l'agence. Expliquer les requêtes SQL suivantes saisies par le gestionnaire :

```
SELECT id_client FROM clients
  WHERE nom = 'Barc' AND prenom = 'Jean' AND date_naissance =
  '1972/06/29' AND pays = 'Allemagne';

SELECT id_reservation FROM reservations
  WHERE id_client = 1243;
```

La première requête permet d'obtenir l'identifiant de M Jean Barc.
La deuxième requête permet d'obtenir l'identifiant de la réservation du client ayant pour identifiant 1243 (c'est à dire M Jean Barc).

8. Écrire les deux requêtes de la question 7. sous la forme d'une requête unique.

```
SELECT id_reservation
FROM reservations
JOIN clients ON reservations.id_client = clients.id_client
WHERE nom = 'Barc' AND prenom = 'Jean' AND date_naissance =
'1972/06/29' AND pays = 'Allemagne'
```

9. Un client souhaite modifier sa réservation d'identifiant 20456. Il souhaite remplacer la croisière de cette réservation par la toute dernière offre de l'agence : la *Croisière Puerto*. Écrire une requête SQL qui permet de mettre à jour la base de données pour lui donner satisfaction.

```
UPDATE reservation
SET nom_croisiere = 'Croisuère Puerto'
WHERE id_reservation = 20456
```

10. Donner une requête SQL permettant d'obtenir les noms, prénoms et dates de naissance des clients ayant choisi la croisière nommée *Croisière Piano* ou celle nommée *Croisière Puerto.*

```
SELECT nom, prenom, date_naissance
FROM clients
JOIN reservations ON clients.id_client = reservations.id_client
WHERE nom_croisiere = 'Croisiere Piano'
OR nom_croisiere = 'Croisiere Puerto'
```