BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

ÉPREUVE BLANCHE

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Mardi 24 janvier 2023

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.

Ce sujet comporte 7 pages numérotées de 1/7 à 7/7 .

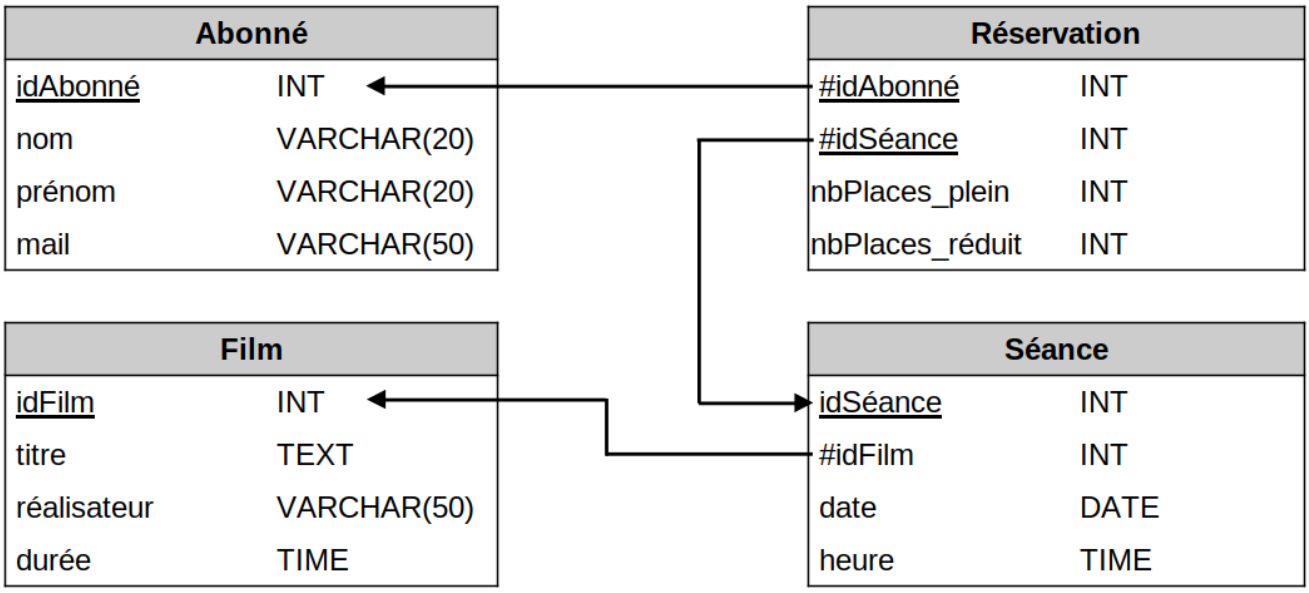
Chaque exercice est noté sur 4 points.

EXERCICE 1

Cet exercice porte sur les bases de données relationnelles et le langage SQL.

Cet exercice utilise les mots du langage SQL suivants : SELECT, FROM, WHERE, JOIN, ON, UPDATE, SET, DELETE, INSERT INTO, COUNT, AND, OR.

Une salle de cinéma propose un site Web à ses abonnés afin d'effectuer des réservations de séances en ligne. Deux tarifs sont proposés : plein et réduit (-16 ans, sénior +65 ans, étudiants, . . .). Le site est associé à une base de données dont le modèle relationnel contient les quatre relations décrites ci-dessous :



Un attribut souligné correspond à une clé primaire et un attribut précédé du symbole # à une clé étrangère.

Voici un extrait de quelques enregistrements des relations `Film`, `Séance` et `Abonné` :

idFilm	titre	réalisateur	durée
1	Le parfum vert	Nicolas Pariser	101
2	Le royaume des étoiles	Ali Samadi Ahadi	85
8	Wakanda forever	Ryan Coogler	132
...	...	...	...

Extrait de la relation `Film` , les durées sont en minutes.

idSéance	idFilm	date	heure
35	1	2023-01-12	21:00
737	8	2023-01-13	18:00
738	8	2023-01-13	21:20

Extrait de la relation `Séance`, les dates sont au format `aaaa-mm-jj`.

idAbonné	nom	prénom	mail
1	Henry	Jean	jean.henry@envoi.fr
2	Jacquin	Emma	jacquin.emma@mail.com
13	Dupont	Charles	cdupont@envoi.fr

Extrait de la relation `Abonné`.

1. **a)** Définir le rôle d'une clé primaire.

   b) Définir le rôle d'une clé étrangère.

   c) Déterminer, en justifiant, si un abonné peut réserver plusieurs fois une même séance.

2. M. Charles Dupont réserve trois places au tarif plein et deux places au tarif réduit pour assister à la projection du film ≪Wakanda forever≫ le 13 janvier 2023 à `18:00`.

   a) À l'aide des extraits des relations donnés précédemment, recopier et compléter l'enregistrement correspondant dans la relation `Réservation` ci-dessous :

idAbonné	idSéance	nbPlaces_plein	nbPlaces_réduit

   b) Écrire en SQL la requête précédente.

3. En SQL, la fonction `COUNT()` permet de compter le nombre d'enregistrements dans une table. Exprimer en langage naturel la requête SQL suivante :

```
SELECT COUNT(*)
FROM Seance
WHERE date = "2023-01-10" OR date = "2023-01-11";
```

4. Écrire en SQL les requêtes permettant d'effectuer les tâches suivantes :

   a) Afficher le nom et le prénom de tous les abonnés.

   b) Afficher le titre et le réalisateur des films de moins de 120 minutes.

   c) Afficher le titre et la durée des films projetés le 14 janvier à `21:00`.

5. **a)** Écrire une requête en SQL permettant de modifier la durée du film ≪ Avatar 2 ≫ à 192 (initialement enregistré avec 132).

   b) On souhaite écrire une requête SQL permettant de supprimer la séance dont l'attribut `idSéance` vaut 135.

   Déterminer la contrainte d'intégrité que pourrait violer cette requête.

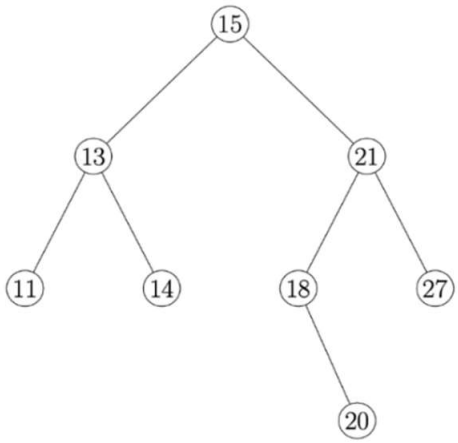
   c) Écrire la requête précédente en SQL.

EXERCICE 2

Cet exercice porte sur les arbres binaires de recherche, la programmation orientée objet et la récursivité.

Dans cet exercice, la taille d'un arbre est le nombre de nœuds qu'il contient. Sa hauteur est le nombre de nœuds du plus long chemin qui joint le nœud racine à l'une des feuilles (nœuds sans sous-arbres). On convient que la hauteur d'un arbre ne contenant qu'un nœud vaut 1 et la hauteur de l'arbre vide vaut 0.

1. On considère l'arbre binaire représenté ci-dessous :



a) Donner la taille de cet arbre.

b) Donner la hauteur de cet arbre.

c) Représenter sur la copie le sous-arbre à droite du nœud de valeur 15.

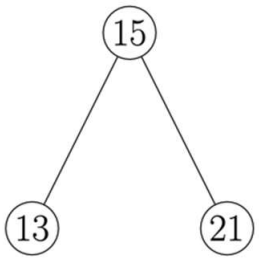
d) Justifier que l'arbre de la figure 1 est un arbre binaire de recherche.

e) On insère la valeur 17 dans l'arbre de la figure 1 de telle sorte que 17 soit une nouvelle feuille de l'arbre et que le nouvel arbre obtenu soit encore un arbre binaire de recherche. Représenter sur la copie ce nouvel arbre.

2. On considère la classe `Noeud` définie de la façon suivante en Python :

```
class Noeud:
    def __init__(self, gauche, valeur, droite):
        self.gauche = gauche
        self.valeur = valeur
        self.droite = droite
```

a) Parmi les trois instructions **(A)**, **(B)** et **(C)** suivantes, écrire sur la copie la lettre correspondant à celle qui construit et stocke dans la variable abr l'arbre représenté ci-dessous.



(A) `abr = Noeud(Noeud(Noeud(None, 13, None), 15, None), 21, None)`

(B) `abr = Noeud(None, 13, Noeud(Noeud(None, 15, None), 21, None))`

(C) `abr = Noeud(Noeud(None, 13, None), 15, Noeud(None, 21, None))`

b) Recopier et compléter la ligne 7 du code de la fonction `inserer` ci-dessous qui prend en paramètres une valeur `v` et un arbre binaire de recherche `abr` et qui renvoie l'arbre obtenu suite à l'insertion de la valeur `v` dans l'arbre `abr`. Les lignes 8 et 9 permettent de ne pas insérer la valeur `v` si celle-ci est déjà présente dans `abr`.

```
1  def inserer(v, abr):
2      if abr is None:
3          return Noeud(None, v, None)
4      if v > abr.valeur:
5          return Noeud(abr.gauche, abr.valeur, inserer(v, abr.droite))
6      elif v < abr.valeur:
7          return ...
8      else:
9          return abr
```

3. La fonction `nb_sup` prend en paramètres une valeur `v` et un arbre binaire de recherche `abr` et renvoie le nombre de valeurs supérieures ou égales à la valeur `v` dans l'arbre `abr`.

Le code de cette fonction `nb_sup` est donné ci-dessous :

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

a) On exécute l'instruction `nb_sup(16, abr)` dans laquelle `abr` est l'arbre **initial** de la question **1.**. Déterminer le nombre d'appels à la fonction `nb_sup`.

b) L'arbre passé en paramètre étant un arbre binaire de recherche, on peut améliorer la fonction `nb_sup` précédente afin de réduire ce nombre d'appels.

Écrire sur la copie le code modifié de cette fonction.

EXERCICE 3

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

Un supermarché met en place un système de passage automatique en caisse. Un client scanne les articles à l'aide d'un scanner de code-barres au fur et à mesure qu'il les ajoute dans son panier.

Les articles s'enregistrent alors dans une structure de données. La structure de données utilisée est une file définie par la classe `Panier`, avec les primitives habituelles sur la structure de file.

Pour faciliter la lecture, le code de la classe `Panier` n'est pas écrit.

```
class Panier():
    def __init__(self):
        "Initialise la file comme une file vide."

    def est_vide(self):
        "Renvoie True si la file est vide, False sinon."

    def enfile(self, e):
        "Ajoute l'element e en derniere position de la file, ne renvoie rien."

    def defile(self):
        "Retire le premier element de la file et le renvoie."
```

Le panier d'un client sera représenté par une file contenant les articles scannés.

Les articles sont représentés par des tuples `(code_barre, designation, prix, horaire_scan)` où

- `code_barre` est un nombre entier identifiant l'article ;
- `designation` est une chaine de caractères qui pourra être affichée sur le ticket de caisse ;
- `prix` est un nombre décimal donnant le prix d'une unité de cet article ;
- `horaire_scan` est un nombre entier de secondes permettant de connaitre l'heure où l'article a été scanné.

1. On souhaite ajouter un article dont le tuple est le suivant `(31002, "café noir", 1.50, 50525)`.

   Écrire le code utilisant une des quatre méthodes ci-dessus permettant d'ajouter l'article à l'objet de classe `Panier` appelé `panier_1`.

2. On souhaite définir une méthode `remplir` de paramètre `panier_temp` dans la classe `Panier` permettant de transférer vers la file tout le contenu d'un autre panier `panier_temp` qui est aussi un objet de type `Panier`.

   Recopier et compléter le code de la méthode `remplir`.

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

3. Pour que le client puisse connaitre à tout moment le montant de son panier, on souhaite ajouter une méthode `prix_total` (sans paramètres) à la classe `Panier` qui renvoie la somme des prix de tous les articles présents dans le panier.

   Écrire le code de la méthode `prix_total`.

   Attention, après l'appel de cette méthode, le panier devra toujours contenir ses articles.

4. Le magasin souhaite connaitre pour chaque client la durée du passage en caisse. Cette durée sera obtenue en faisant la différence entre le champ `horaire_scan` du dernier article scanné et le champ `horaire_scan` du premier article scanné dans le panier du client. Un panier vide renverra une durée égale à `None`. On pourra accepter que le panier soit vide après l'appel de cette méthode.

Écrire une méthode `duree_passage_en_caisse` de la classe `Panier` qui renvoie cette durée.