MÉTROPOLE – 2026 – sujet1

EXERCICE 3 (8 POINTS)

Cet exercice porte sur les arbres, l'algorithmique des arbres, et les bases de données.

On s'intéresse au fonctionnement interne d'une plateforme en ligne dédiée spécifiquement aux débats. Plutôt que de lister des messages des utilisateurs dans l'ordre chronologique, les différentes contributions sont structurées de façon arborescente afin de mieux comprendre les liens entre les arguments. Chaque contribution vient se rattacher à une affirmation, soit afin d'appuyer cette affirmation avec un argument pour, soit afin de l'attaquer avec un argument contre.

Cet exercice contient deux parties indépendantes l'une de l'autre.

PARTIE A : STRUCTURE ARBORESCENTE DES ARGUMENTS

Dans cette partie on modélise un arbre de débat pour un sujet de débat. La figure 1 montre un exemple d'arbre de débat.

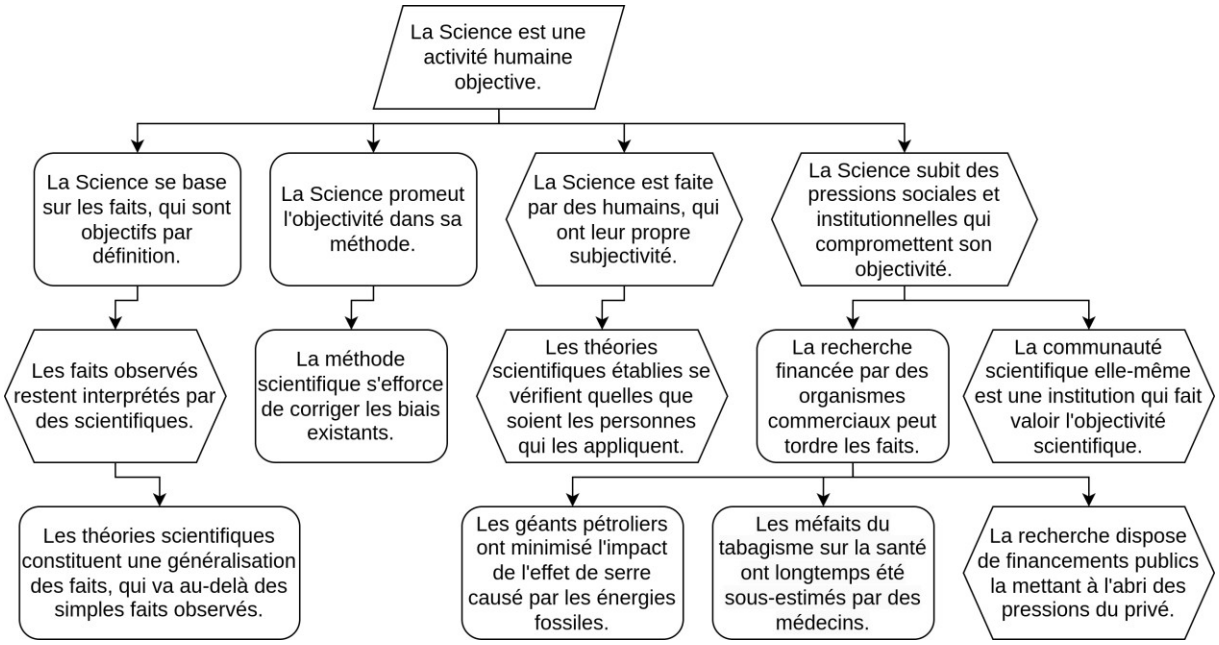

Figure 1. Arbre de débat sur l'objectivité de la science

Dans un arbre de débat, chaque nœud est une affirmation, et il y en a de trois sortes :

- le sujet du débat (dans le parallélogramme) est une affirmation, c'est le point de départ du débat ;
- un argument pour (dans un rectangle aux coins arrondis) peut venir soutenir une affirmation, cet argument est lui-même une affirmation ;
- un argument contre (dans un hexagone) peut venir attaquer une affirmation, cet argument est lui-même une affirmation.

De plus, chaque affirmation peut être likée (j'aime) ou dislikée (je n'aime pas) par les utilisateurs.

1. Indiquer le nom donné au nœud « point de départ du débat » dans le vocabulaire des arbres.

2. Indiquer en justifiant si un arbre de débat est un arbre binaire ou non.

On modélise la structure d'arbre de débat à l'aide de la classe suivante :

```
1  class Affirmation:
2      def __init__(self, phrase):
3          """
4          Création d'un objet Affirmation à partir d'une
5          phrase (str).
6          """
7          self.contenu = phrase
8          self.sorte = "" # vaudra "sujet", "pour" ou "contre"
9                          # vaut "" avant ajout dans l'arbre
10         self.arguments = []
11         self.nb_likes = 0
12         self.nb_dislikes = 0
```

3. Donner le nom et le type de chaque attribut de la classe `Affirmation`.

On souhaite ajouter à cette classe une méthode `soutenir` qui permet d'ajouter un argument pour à l'affirmation.

On suppose à disposition une méthode `contrer` similaire qui permet d'ajouter un argument contre à l'affirmation.

```
1  def soutenir(self, argument):
2      """
3      Ajoute l'Affirmation `argument` comme argument pour de
4      l'Affirmation `self`.
5
6      Précondition : l'Affirmation ne doit pas avoir déjà été
7      utilisée, ni comme sujet, ni comme pour, ni comme
8      contre, ce qu'on vérifie grâce à son attribut `sorte`.
9      """
10     assert argument.sorte == ...
11     ... .append(argument)
12     argument.sorte = "..."
```

4. Recopier et compléter les lignes 10 à 12 de la méthode `soutenir`, en respectant sa documentation.

Pour mesurer les débats les plus vifs, la plateforme de débats dispose de plusieurs mesures.

5. Écrire la méthode `nb_contre` qui permet d'obtenir le nombre d'arguments contre directement rattachés à cette `Affirmation`.

On s'intéresse à la méthode `mystere` suivante, qui utilise la méthode `nb_contre` :

```
1  def mystere(self):
2      m = self.nb_contre()
3      for arg in self.arguments:
4          candidat = arg.mystere()
5          if candidat > m:
6              m = candidat
7      return m
```

6. Expliquer par une phrase ce que renvoie la méthode `mystere`.

Afin de trancher le débat dans sa globalité, on souhaite évaluer une `Affirmation`, en tenant compte de la totalité des likes et des dislikes de chaque argument, et en tenant compte du fait que ces arguments soient pour ou contre. Si l'évaluation est positive le pour l'emporte, si elle est négative le contre l'emporte.

La formule pour l'évaluation d'une affirmation correspond à la somme de ses likes et des évaluations de ses arguments pour, à laquelle on soustrait ses dislikes et les évaluations de ses arguments contre.

7. Écrire, en utilisant uniquement des lignes de code choisies parmi les lignes suivantes, une méthode `evaluation` qui renvoie le résultat du calcul décrit dans la formule ci-dessus. Les lignes ne sont pas toutes à utiliser et l'indentation est à adapter.

```
def evaluation(self):
def evaluation():
total = self.nb_likes + self.nb_dislikes
total = self.nb_likes - self.nb_dislikes
total = self.nb_dislikes - self.nb_likes
total = self.nb_likes * self.nb_dislikes
for arg in self.arguments:
for arg in self.arguments_pour:
for arg in self.arguments_contre:
if arg.sorte == "sujet":
if arg.sorte == "pour":
if arg.sorte == "contre":
else:
total = arg.evaluation()
total = total + arg.evaluation()
total = total - arg.evaluation()
total = total * arg.evaluation()
return total
return 1 + total
```

PARTIE B : BASE DE DONNÉES DES UTILISATEURS ET DE LEURS CONTRIBUTIONS

Dans cette partie, on pourra utiliser les clauses du langage SQL pour :

- construire des requêtes d'interrogation à l'aide de `SELECT`, `FROM`, `WHERE` (avec les opérateurs logiques `AND`, `OR`) et `JOIN ... ON` ;
- construire des requêtes d'insertion et de mise à jour à l'aide de `UPDATE`, `INSERT` et `DELETE`.

Pour manipuler toutes les informations sur plusieurs sujets de débats, la plateforme dispose d'un Système de Gestion de Bases de Données (SGBD).

8. Parmi les propositions suivantes, indiquer celles qui font partie des rôles d'un SGBD :

   a) sécuriser les accès à la base de données ;

   b) assurer l'alimentation électrique des serveurs ;

   c) assurer la persistance des données même en cas de panne matérielle ;

   d) gérer les accès en parallèle de plusieurs utilisateurs ;

   e) assurer des connexions en HTTPS au serveur.

Pour gérer les données, le schéma relationnel utilisé est représenté sur la figure 2.

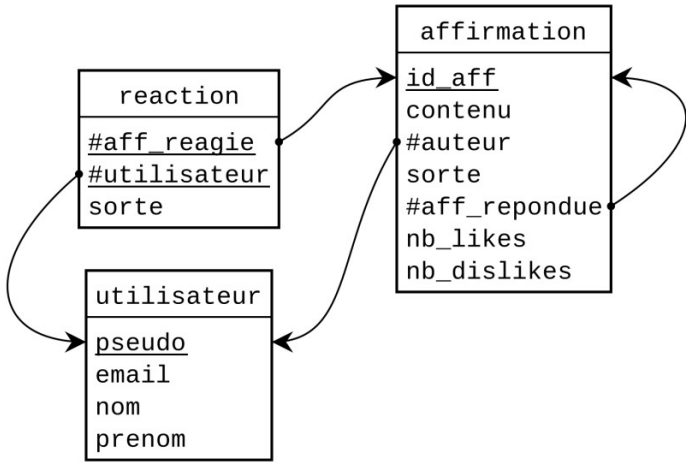

Figure 2. Schéma relationnel de la plateforme de débat

La clé primaire de chaque table correspond à l'ensemble de ses attributs soulignés. Chaque attribut précédé d'un « # » est une clé étrangère.

L'attribut `sorte` de la table `affirmation` peut valoir `'sujet'` quand c'est l'affirmation initiale d'un sujet de débat, ou bien `'pour'` ou `'contre'` quand c'est une réponse à une autre affirmation (référencée par l'attribut `aff_repondue`). L'attribut `contenu` correspond à la phrase elle-même de l'affirmation.

L'attribut `sorte` de la table `reaction` peut valoir `'like'` ou `'dislike'`, et s'applique à l'affirmation référencée par l'attribut `aff_reagie`.

Voici des exemples d'extraits des tables `affirmation` et `reaction`. On a omis, pour ces exemples, les attributs `auteur` et `contenu` de la table `affirmation` :

affirmation				
id_aff	**sorte**	**aff_repondue**	**nb_likes**	**nb_dislikes**
0	'sujet'	NULL	42	17
1	'pour'	0	20	20
2	'pour'	0	40	10
3	'contre'	0	13	12
4	'contre'	0	18	6
7	'contre'	1	15	16

reaction		
aff_reagie	**utilisateur**	**sorte**
0	'alice'	'like'
1	'alice'	'dislike'
0	'bob'	'dislike'
1	'bob'	'dislike'

9. Indiquer en justifiant quel autre attribut de la table `utilisateur` aurait aussi pu servir de clé primaire.

On rappelle que la fonction d'agrégation `COUNT` permet de compter les éléments du résultat d'une requête en plaçant `COUNT(*)` dans la clause `SELECT`.

10. Écrire une requête SQL permettant d'obtenir le nombre d'affirmations qui ont 50 likes ou plus.

11. Écrire une requête qui permet d'obtenir les contenus des sujets et leur nombre de likes, créés par les utilisateurs dont les prénom et nom sont respectivement `'Pierre'` et `'Durand'`.

Le mot-clé AS permet de donner un autre nom à une table au sein d'une requête. Il est notamment nécessaire lorsqu'on a besoin de faire la jointure d'une table avec elle-même. On considère la requête SQL suivante :

```
SELECT aff.contenu, rep.contenu
FROM affirmation AS aff JOIN affirmation AS rep
  ON rep.aff_repondue = aff.id_aff
WHERE rep.nb_likes >= 2 * aff.nb_likes
  AND rep.sorte = 'contre'
```

12. Donner une interprétation en français de ce que permet d'obtenir la requête ci-dessus.

L'utilisateur au pseudo `'i<3descartes'` approuve par un like l'affirmation `'Cogito ergo sum'` qui a l'`id_aff` 108.

13. Recopier et compléter les deux requêtes SQL suivantes, permettant de mettre à jour les informations de la base de données suite à cette action.

```
... reaction
VALUES (..., ..., ...);

... affirmation
SET nb_likes = ...
WHERE ... = ...;
```

L'utilisateur au pseudo `'i<3rgpd'` souhaite faire valoir son droit à l'effacement, et supprimer son compte utilisateur et toutes les données qui y sont liées. Dans cette situation, la plateforme de débat, qui souhaite maintenir la qualité des échanges, suit la procédure suivante :

- elle anonymise toutes les affirmations de l'utilisateur (en remplaçant l'attribut `auteur` par la valeur `NULL`), considérant que les affirmations elles-mêmes ne sont pas des données personnelles ;
- elle supprime toutes les réactions de l'utilisateur (mais sans toucher aux nombres de likes et de dislikes des affirmations correspondantes) ;
- et enfin elle supprime l'utilisateur lui-même.

14. Expliquer pourquoi il est nécessaire de supprimer les réactions de l'utilisateur avant de supprimer l'utilisateur.

15. Écrire les trois requêtes qui permettent la suppression des données du compte de `'i<3rgpd'` en suivant la procédure décrite ci-dessus.