Lycée Les 3 Sources **Bourg-Lès-Valence**
NSI T^le Année **2024-25**

Devoir surveillé n°4

Nom et prénom :

Exercice 1 : (16pt) *Cet exercice porte sur la programmation Python, la programmation orientée objet, les bases de données relationnelles et les requêtes SQL.*

Partie A

Une entreprise, présente sur différents sites en France, attribue à chacun de ses employés un numéro de badge unique.
Dans le tableau ci-dessous, on donne le numéro de badge, le nom, le prénom et les années de naissance et d'entrée dans l'entreprise de quelques salariés.

numéro badge	nom	prénom	année de naissance	année d'entrée
112	LESIEUR	Isabelle	1982	2005
2122	VASSEUR	Adrien	1962	1980
135	HADJI	Hakim	1992	2015

Pour chaque personne, on souhaite stocker les informations dans un objet de la classe `Personne` définie ci-dessous :

```
class Personne():
    def __init__(self, num, n, p, a_naiss, a_entree):
        self.num_badge = num
        self.nom = n
        self.prenom = p
        self.annee_naissance = a_naiss
        self.annee_entree = a_entree
```

1) Écrire à l'aide du tableau précédent, l'instruction permettant de créer l'objet `personneA` de la première personne du tableau : LESIEUR Isabelle.

2) Donner l'instruction permettant d'obtenir le numéro de badge de l'objet `personneA` instancié à la question précédente.

On souhaite ajouter une méthode `annee_anciennete` à la classe Personne qui donne le nombre d'années d'ancienneté d'une personne au sein de l'entreprise. Par exemple : Madame LESIEUR Isabelle a une ancienneté dans l'entreprise de 19 ans en considérant que nous sommes en 2024.

3) Compléter le code suivant de la méthode `annee_anciennete` :

```
    def annee_anciennete(self):
        return ...
```

On considère la classe `Personnel` qui modélise la liste du personnel d'une entreprise et dont le début de l'implémentation est la suivante :

```
class Personnel:
    def __init__(self):
        self.liste = []
```

4) Écrire la méthode `ajouter` permettant d'ajouter un objet de type `Personne` à la liste du personnel de l'entreprise de la classe `Personnel`.

5) Écrire la méthode `effectif` de la classe `Personnel`. Cette méthode devra renvoyer le nombre de personnes présentes dans l'entreprise.

6) Compléter la méthode `donne_nom` de la classe `Personnel`. Cette méthode prend en paramètre le numéro de badge d'une personne et renvoie le nom de la personne correspondant à ce badge si elle existe, ou **`None`** sinon.

```
    def donne_nom(............, num):
        for elt in self.liste:
            if num == ............:
                return ............
        return ............
```

7) Lors de la célèbre cérémonie des vœux, l'entreprise souhaite mettre à l'honneur les personnes ayant exactement 10 ans d'ancienneté dans l'entreprise. Écrire une méthode de la classe `Personnel` `nb_personne_honneur` qui prend en paramètre l'année de la cérémonie et qui retourne le nombre de personne(s) à mettre à l'honneur.

8) Écrire une méthode `plus_anciens` de la classe `Personnel` qui retourne la liste des numéros de badge des personnes ayant la plus grande ancienneté dans l'entreprise.

Partie B

On utilise maintenant une base de données relationnelle. La table `Personnel` dont un extrait est donné ci-dessous contient toutes les données importantes sur le personnel de l'entreprise. L'attribut `num_centre` désigne le numéro du centre dans lequel travaille une personne.

Table `Personnel`

num_badge	nom	prenom	num_centre	annee_naiss	annee_debut
112	LESIEUR	Isabelle	1	1982	2005
2122	VASSEUR	Adrien	2	1962	1980
135	HADJI	Hakim	1	1992	2015

L'attribut `num_badge` est la clé primaire pour la table `Personnel`.

9) Décrire par une phrase en français le résultat de la requête SQL suivante :

```
SELECT nom, prenom
FROM Personnel
WHERE num_centre = 2;
```

10) Monsieur HADJI Hakim vient d'obtenir une mutation pour le centre numéro 3. Donner la requête permettant de modifier son numéro de centre sachant que son numéro de badge est 135.

On souhaite proposer plus d'informations sur les différents centres de l'entreprise. Pour cela, on crée une deuxième table `Centre` avec les attributs suivants :

- `num` de type INT ;
- `nom` de type TEXT ;
- `num_tel` de type TEXT ;
- `ville` de type TEXT.

Table `Centre`

num	nom	num_tel	ville
1	Normandie	0450646859	Caen
2	PACA	0450646859	Marseille

11) Expliquer l'intérêt d'utiliser deux tables (`Personnel` et `Centre`) au lieu de regrouper toutes les informations dans une seule table.

12) Expliquer comment les tables `Centre` et `Personnel` sont mises en relation.

13) Écrire une requête permettant d'avoir les noms des personnes travaillant dans le centre de Lille et ayant été embauchées entre 2015 (inclus) et 2020 (inclus).

Le centre de Normandie vient d'être fermé, mais les personnes de ce centre n'ont pas encore été affectées dans leur nouveau centre. On souhaite mettre à jour la table `Centre` en premier à l'aide de la requête suivante.

14) Expliquer pourquoi cette requête a renvoyé une erreur.

```
DELETE *
FROM Centre
WHERE nom = 'Normandie';
```

Exercice 2 : (14pt) *Cet exercice porte sur le codage binaire, les bases de données relationnelles et les requêtes SQL.*

Cet exercice est composé de deux parties peu dépendantes entre elles. Lorsqu'il y a un accident, les pompiers essaient d'intervenir sur les lieux le plus rapidement possible avec les équipes et le matériel adéquats. On s'intéresse à l'étude d'un système informatique simplifié permettant de répondre à certaines de leurs problématiques.

Chaque pompier possède des aptitudes opérationnelles qui lui permettent de tenir un rôle. Lors du départ d'un véhicule (on parle d'agrès), il faut a minima un conducteur et un chef d'agrès.

Pour simplifier, on considère que

- un Véhicule Tout Usage (VTU) ne requiert que le duo conducteur et chef d'agrès, donc deux pompiers ;
- un Véhicule de Secours et d'Assistance aux Victimes (VSAV) requiert, en plus du duo conducteur et chef d'agrès, un équipier, donc trois pompiers ;
- un Fourgon Pompe Tonne (FPT) requiert, en plus du duo conducteur et chef d'agrès, un chef d'équipe et un équipier, donc quatre pompiers.

On souhaite entrer dans une base de données l'ensemble de ces informations afin de pouvoir conserver un historique des interventions.

Partie A – Encodage binaire

Afin de gagner de la place mémoire, on décide de coder l'ensemble des aptitudes sur un entier de 8 bits plutôt que d'écrire en toutes lettres "équipier", "chef d'équipe", etc. Cet ensemble d'aptitudes formera la qualification du pompier considéré.

- Un personnel non formé est codé par 0 ;
- l'aptitude "équipier" est codée par le bit de rang 0 (celui de poids le plus faible), soit 2^0 ;
- l'aptitude "chef d'équipe" est codée par le bit de rang 1, soit 2^1 ;
- l'aptitude "chef d'agrès" est codée par le bit de rang 2, soit 2^2 ;
- enfin, l'aptitude "conducteur" est codée par le bit de rang 3, soit 2^3.

Remarque : un chef d'équipe étant nécessairement équipier, on lui ajoute cette aptitude dans son codage. De même, un chef d'agrès est nécessairement un chef d'équipe et un équipier : on lui ajoute ces aptitudes dans son codage.

1) Justifier que la qualification décimale 11 correspond à un chef d'équipe conducteur.

2) Déterminer le codage décimal de la qualification chef d'agrès conducteur.

3) Expliquer pourquoi dans la situation décrite, un pompier ne peut pas avoir de qualification dont le codage décimale est 4.

4) Avec ce codage sur un octet, indiquer combien de nouvelles aptitudes peuvent être définies.

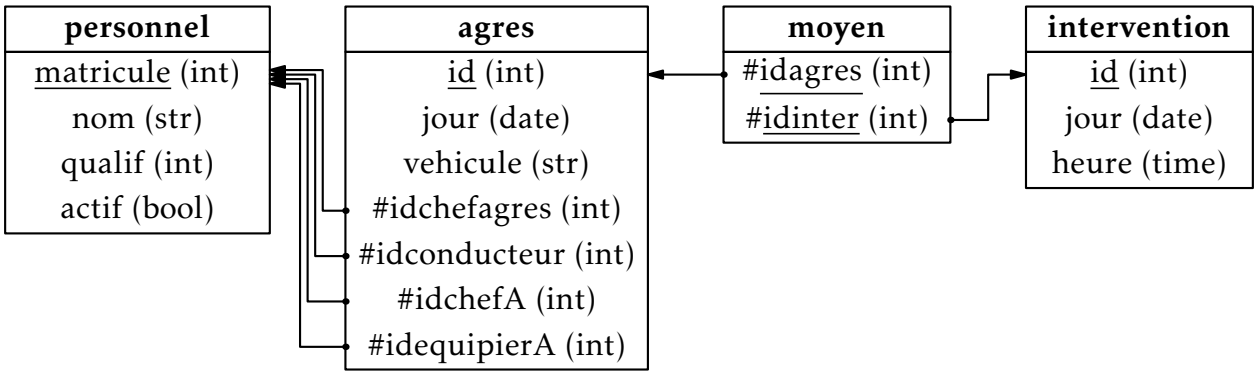
Nom et prénom :

On considère que chacune des quatre aptitudes aurait pu être codée par une chaîne de 10 caractères dont chaque caractère utilise 1 octet en mémoire.

5) Choisir, avec justification, l'économie mémoire que le codage sur un entier de 8 bits permet de faire comparée au codage basé sur les chaînes de caractères : environ 10 %, 30 %, 50 %, 98 %.

Partie B

On considère maintenant la base de données relationnelle suivante :



La table `personnel` est composée :

- du matricule unique du pompier ;
- de son nom ;
- de sa qualification (selon le codage vu avant) ;
- d'un attribut qui précise s'il est actif (1) ou inactif (0).

personnel			
matricule	nom	qualif	actif
10	'Sam'	3	1
16	'Charlot'	1	0
31	'Red'	23	0
83	'Vaillante'	7	1
2501	'Marco'	1	1
2674	'Aicha'	23	1
3004	'Fatou'	7	1
4044	'Abdel'	19	1
4671	'Mamadou'	17	1
5301	'Zoe'	17	1
7450	'Medhi'	3	1
8641	'Gaia'	1	1
8678	'Kevin'	17	0
8682	'Marie'	1	1
9153	'Fred'	23	1

La table `agres` est composée :

- de son identifiant ;
- du jour où l'agrès se tient prêt à intervenir ;
- du type de véhicule ;
- de l'identifiant du chef d'agrès ;
- de l'identifiant du conducteur ;
- de l'identifiant du chef d'équipe si le véhicule le nécessite, sinon le champ est à `NULL` ;
- de l'identifiant de l'équipier si le véhicule le nécessite, sinon le champ est à `NULL`.

La table `moyen` est composée :

- de l'identifiant de l'agrès appelé sur intervention ;
- de l'identifiant de l'intervention.

La table `intervention` est composée :

- de son identifiant ;
- du jour de début d'intervention (sauf pour les longues interventions où il correspond au jour de l'agrès) ;
- de l'heure de début d'intervention.

On rappelle que `COUNT(*)` permet de compter le nombre de lignes extraites lors d'une requête. Par exemple, pour afficher le nombre de personnes dans la table personnel, on exécute la requête :

```
SELECT COUNT(*) FROM personnel;
```

`DISTINCT` permet de retirer les doublons des réponses. Par exemple, pour afficher tous les noms distincts de la table personnel, on exécute la requête :

```
SELECT DISTINCT(nom) FROM personnel;
```

On considère l'extrait de la base de données sur la page précédente et ci-dessous :

moyen	
idagres	idinter
0	0
2	3
2	4
3	4
4	4
9	5
17	6
22	7
23	8
24	8
24	9

intervention		
id	jour	heure
0	'2023-11-21'	'12:32:21'
1	'2023-11-22'	'22:20:00'
2	'2023-12-17'	'23:17:30'
3	'2024-02-15'	'01:44:06'
4	'2024-02-15'	'12:15:00'
5	'2024-03-02'	'04:58:12'
6	'2024-03-27'	'13:07:18'
7	'2024-05-31'	'05:17:12'
8	'2024-06-11'	'05:38:17'
9	'2024-06-11'	'15:08:56'
10	'2024-06-18''	'07:42:33'

agres						
id	jour	vehicule	idchefagres	idconducteur	idchefA	idequipierA
0	'2023-11-21'	'VSAV'	83	9153	NULL	10
2	'2024-02-15'	'VSAV'	2674	4044	NULL	8641
3	'2024-02-15'	'FPT'	9153	5301	8682	2501
4	'2024-02-15'	'VSAV'	83	4671	NULL	7450
7	'2024-02-29'	'VSAV'	9153	3004	NULL	2501
9	'2024-03-02'	'FPT'	2674	5301	8682	8641
12	'2024-03-21'	'VTU'	3004	5301	NULL	NULL
17	'2024-03-27'	'VSAV'	3004	8682	NULL	10
18	'2024-03-27'	'VSAV'	9153	5301	NULL	10
22	'2024-05-31'	'FPT'	9153	4044	7450	8641
23	'2024-06-11'	'VTU'	83	2674	NULL	NULL
24	'2024-06-11'	'VSAV'	3004	4044	NULL	7450

6) Expliquer la différence entre une clé primaire et une clé étrangère.

7) Expliquer pourquoi la requête suivante génère une erreur pour l'extrait de données.

```
INSERT INTO moyen (idagres, idinter) VALUES (1, 5);
```

8) Proposer une requête SQL qui met à jour l'heure de l'intervention du « 15 février 2024 de 01 heure 44 minutes et 06 secondes » à « 10 heures 44 minutes et 06 secondes ».

9) Préciser le résultat de la requête suivante pour l'extrait de données.

```sql
SELECT nom FROM personnel WHERE actif = 0;
```

10) Proposer une requête SQL qui permet d'afficher les noms des personnels conducteurs actifs. On notera qu'un conducteur possède un attribut `qualif` supérieur ou égal à 16.

11) Écrire l'affichage obtenu après exécution des deux requêtes ci-dessous sur l'extrait de la base de données. Expliquer ce que chacune des requêtes affiche en général.

Requête A

```sql
SELECT COUNT(*) FROM agres
WHERE jour = '2024-03-27';
```

Requête B

```sql
SELECT COUNT(*) FROM moyen AS m
INNER JOIN agres AS a ON a.id = m.idagres
WHERE a.jour = '2024-03-27';
```

12) Proposer une requête qui renvoie sans répétition tous les noms des chefs d'agrès assignés à un véhicule le 15 février 2024.

13) Proposer une requête qui renvoie sans répétition tous les noms des chefs d'agrès engagés en intervention le 11 juin 2024.