ANTILLES-GUYANE – 2026 – sujet2

EXERCICE 1 (6 POINTS)

Cet exercice porte sur les bases de données relationnelles, les requêtes SQL et la programmation objet. Cet exercice comporte deux parties indépendantes.

PARTIE A

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` et `OR`) et `JOIN ... ON` ;
- construire des requêtes d'insertion et de mise à jour à l'aide de `UPDATE`, `INSERT` et `DELETE` ;
- faire des alias avec `AS` ;
- affiner les recherches à l'aide de `DISTINCT` et `ORDER BY` ;
- réaliser des agrégations à l'aide de `COUNT`, `AVG`, `MAX`, `MIN` et `SUM`.

Une agence de location d'espaces de travail (coworking) a créé une base de données comprenant les relations suivantes :

- `salle (id_salle, intitule, type, nb_places, tarif_jour)`
- `reservation (id, id_client, id_salle, date, duree)`
- `client (id_client, societe, num_tel)`

On donne ci-dessous des extraits des relations précédentes :

salle				
id_salle	**intitule**	**type**	**nb_places**	**tarif_jour**
1	B1	Bureau	3	140
2	Ravel	Amphi	90	2300
3	RA	Réunion	10	350
4	B2	Bureau	4	155
5	RB	Réunion	8	300

reservation				
id	**id_client**	**id_salle**	**date**	**duree**
1	2	3	2024-03-24	1
2	1	1	2023-12-17	1
3	2	3	2024-04-16	2
4	1	4	2024-02-05	1

client		
id_client	**societe**	**num_tel**
1	Dupont&co.	0812121212
2	FleursJ	0950505050

L'attribut `id_salle` de la relation `salle`, l'attribut `id` de la relation `reservation` et l'attribut `id_client` de la relation `client` sont des clés primaires.

1. Donner, en justifiant, les clés étrangères de la relation `reservation`.

2. Donner le résultat de la requête suivante, quand on l'applique aux extraits donnés des tables.

```
SELECT intitule FROM salle WHERE nb_places > 5
     ORDER BY tarif_jour;
```

3. Écrire une requête permettant d'obtenir la liste sans doublon des `id_salle` des salles réservées depuis janvier 2024. On pourra utiliser le comparateur > pour les dates.

Un nouveau client de nom de société 'EN' et de numéro de téléphone '0144624055' a réservé le bureau d'identifiant 4 pour une journée le 12 juin 2024.

4. Écrire une suite de requêtes permettant d'ajouter ce client et sa réservation. On attribuera la valeur 3 à `id_client` et la valeur 5 à `id`.

La salle 'RAVEL' a été modifiée, elle peut maintenant contenir 95 personnes.

5. Écrire la requête permettant d'enregistrer cette modification dans la base de données précédente.

6. Écrire une requête permettant d'obtenir le temps total de réservation de la salle d'identifiant 6.

7. Écrire une requête permettant d'obtenir la liste sans doublon des noms de société ayant réservé une salle de réunion.

PARTIE B

On implémente maintenant ces données sous la forme d'une classe en Python donnée ci-dessous. Pour simplifier le code, on considère qu'il n'y a plus de date de réservation mais simplement un état de disponibilité. Lorsqu'une salle est disponible, l'attribut `occupant` prend pour valeur la chaîne vide.

```
class Salle:
    def __init__(self, intitule, type_salle, nb_places,
                 tarif_jour):
        self.intitule = intitule
        self.type_salle = type_salle
        self.nb_places = nb_places
        self.tarif = tarif_jour
        self.dispo = True
        self.occupant = ""
    def reserver(self, client):
        if self.dispo:
            self.dispo = False
            self.occupant = client
            return self.tarif
        return False
```

8. Donner un exemple d'attribut et un exemple de méthode de la classe `Salle`.

9. Écrire la méthode `liberer` qui modifie les attributs d'une salle de manière à la rendre disponible et sans occupant (chaîne de caractères vide).

Le code suivant crée et modifie une liste de salles :

```
S1 = Salle("B1", "Bureau", 3, 140)
S2 = Salle("RAVEL", "Amphi", 90, 2300)
S3 = Salle("RA", "Réunion", 10, 350)
S4 = Salle("B2", "Bureau", 4, 155)
S5 = Salle("RB", "Réunion", 8, 300)
S1.reserver("Dupont&co")
S2.reserver("FleursJ")
liste_salles = [S1,S2,S3,S4,S5]
```

10. Recopier et compléter les lignes 3, 4 et 5 du code de la fonction `salles_dispos` qui prend en paramètre une liste de salles et renvoie la liste des intitulés de celles qui sont disponibles.

```
def salles_dispos(liste_salles):
    res = []
    for ...:
        if ...:
            res.append(...)
    return res
```

On crée maintenant une classe `Seminaire` ayant pour attributs une liste de salles réservées, un prix total et un nom de client.

```
class Seminaire:
    def __init__(self, client):
        self.liste_salles = []
        self.prix_total = 0
        self.client = client
    def ajouter_reservation(self, salle):
        if salle.dispo:
            ...
            ...
            return True
        return False
```

11. Recopier et compléter la méthode `ajouter_reservation` qui ajoute `salle` à la liste des salles réservées, puis réserve cette salle en modifiant le prix total. Elle renvoie `True` si la salle a pu être réservée car elle était libre, et `False` sinon.