BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2023

NUMÉRIQUE ET SCIENCES INFORMATIQUES

JOUR 1

Durée de l'épreuve : **3 heures 30**

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

Le candidat traite les trois exercices proposés.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.
Ce sujet comporte 12 pages numérotées de 1/12 à 12/12.

Exercice 1 — 4 points

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

L'énoncé de cet exercice utilise les mots clefs du langage SQL suivants : `SELECT`, `FROM`, `WHERE`, `JOIN ON`, `UPDATE`, `SET`, `INSERT INTO VALUES`, `COUNT`, `ORDER BY`.

La ligue féminine de basket-ball publie les données relatives à chaque saison sur le site web de la ligue. On y retrouve des informations concernant les équipes participantes, les calendriers et les résultats des match ainsi que les statistiques des joueuses. Dans cet exercice, nous allons nous intéresser à la base de données relationnelle LFP_2021_2022 permettant le stockage et la gestion des données de la saison régulière de basket-ball féminin 2021-2022.

1. Voici ci-dessous le contenu entier de la relation (table) `Equipe` :

id_equipe	nom	adresse	telephone
1	Saint-Amand	39 avenue du Clos, 59230 Saint-Amand-les-Eaux	03 04 05 06 07
2	Basket Landes	15 place Saint-Roch, 40000 Mont-De-Marsan	05 06 07 08 09
3	Villeneuve d'Ascq	2 rue Breughel, 59650 Villeneuve-d'Ascq	03 02 01 00 01
4	Tarbe	Quai de l'Adour, 65000 Tarbes	05 04 03 02 02
5	Lyon	451 cours d'Emile Zola, 69100 Villeurbanne	04 05 06 07 08
6	Bourges	6 rue du Pré Doulet, 18000 Bourges	02 03 04 05 06
7	Charleville-Mézières	Rue de la Vieille Meuse, 08000 Charleville-Mézières	03 05 07 09 01
8	Landerneau	Kerouel, 29410 Pleyber-Christ	02 04 06 08 00
9	Angers	330 rue Saint-Léonard, 49000 Angers	02 00 08 06 04
10	Lattes Montpellier	157 rue de la Porte Lombarde, 34970 Lattes	04 03 02 01 00
11	Charnay	Allée des Ecoliers, 71850 Charnay-lès-Mâcon	03 01 09 07 05
12	Roche Vendée	BP 151, 85004 La Roche-Sur-Yon Cedex	02 05 08 01 04

On donne ci-contre le schéma relationnel de la table `Equipe`.
Dans ce schéma, un attribut souligné indique qu'il s'agit d'une clé primaire.

`Equipe`

id_equipe	INT
nom	VARCHAR(50)
adresse	VARCHAR(100)
telephone	VARCHAR(20)

a) Après le chargement de la table `Equipe`, expliquer pourquoi la requête suivante produit une erreur :

```
INSERT INTO Equipe
VALUES (11, "Toulouse", "2 rue du Nord, 40100 Dax", "05 04 03 02 01");
```

La requête produit une erreur car l'attribut `id_equipe` de valeur `11` est déjà dans la relation `Equipe`. Cet attribut doit être unique car c'est la clé primaire de la relation : c'est la contrainte d'unicité.

b) Expliquer le choix du domaine de l'attribut `telephone`.

Le domaine `VARCHAR` est utilisé pour le numéro de téléphone pour conserver les espaces entre chaque paire de chiffres. Si le domaine `INT` avait été utilisé, les espaces seraient supprimés.

c) Donner le résultat de la requête suivante :

```
SELECT nom, adresse, telephone FROM Equipe WHERE id_equipe = 5;
```

Cette requête renvoie :

```
Lyon    451 cours Emile Zola, 69100 Villeurbanne    04 05 06 07 08
```

d) Donner et expliquer le résultat de la requête suivante :

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

Cette requête renvoie 12. Elle renvoie le nombre d'entités dans la relation `Equipe`.

e) Écrire la requête SQL permettant d'afficher les noms des équipes par ordre alphabétique.
Correction :

```
SELECT nom FROM Equipe
ORDER BY nom ASC ;
```

f) Écrire la requête SQL permettant de corriger le nom de l'équipe dont l'`id_equipe` est égal à 4. Le nom correct est "Tarbes".
Correction :

```
UPDATE Equipe
SET nom = "Tarbes"
WHERE id_equipe = 4 ;
```

2. Sur le site web de la fédération de basket-ball féminin, nous pouvons consulter la composition des équipes. Pour chaque joueuse, on peut y lire en plus de son nom, sa date de naissance, sa taille ainsi que le poste occupé dans l'équipe. Ces informations sont présentées dans une page web dont le titre est « Fiche Joueuse », page construite à partir de la table `Joueuse` dont voici un extrait :

id_joueuse	nom	prenom	date_naissance	taille	poste	id_equipe
1	Berkani	Lisa	19/05/1997	176	2	7
2	Alexander	Kayla	05/01/1991	193	5	5
3	Magarity	Regan	30/04/1996	192	4	2
4	Muzet	Johanna	08/07/1997	183	3	11
5	Kalu	Ezinne	26/06/1992	173	2	8
6	Sigmundova	Jodie Cornelie	20/04/1993	193	5	9
7	Dumerc	Céline	09/07/1982	162	2	2
8	Slonjsak	Iva	16/04/1997	183	3	9
9	Michel	Sarah	10/01/1989	180	2	6
10	Lithard	Pauline	11/02/1994	164	1	1

On donne ci-contre le schéma relationnel de la table `Joueuse`.
Un attribut souligné indique qu'il s'agit d'une clé primaire. Le symbole # devant un attribut indique qu'il s'agit d'une clé étrangère.
La clé étrangère `Joueuse.id_equipe` fait référence à la clé primaire `Equipe.id_equipe` de la table `Equipe`.

`Joueuse`

id_joueuse	INT
nom	VARCHAR(50)
prenom	VARCHAR(50)
date_naissance	DATE
taille	INT
poste	INT
#id_equipe	INT

a) Expliquer pourquoi l'attribut `id_equipe` a été déclaré clé étrangère.

L'attribut `id_equipe` a été déclarée clé étrangère de la relation `Joueuse` afin de faire référence aà l'attribut `id_equipe` de la relation `Equipe`. Une entité de la relation `Joueuse` doit donc être liée à une entité de la relation `Equipe`.

b) On souhaite supprimer toutes les informations relatives à une équipe. Expliquer pourquoi on ne peut pas directement supprimer cette équipe dans la table `Equipe`.

On ne peut pas supprimer directement l'équipe dans la relation `Equipe` car certaines entités de la relation `Joueuse` font référence à cette équipe : c'est la contrainte de référence.

Il faudrait d'abord supprimer toutes les entités de la relation `Joueuse` qui font référence à cette équipe avant de supprimer l'équipe dans la relation `Equipe`.

c) Écrire la requête SQL qui permet d'afficher les noms et les prénoms des joueuses de l'équipe d'Angers par ordre alphabétique des noms. On supposera que l'utilisateur qui écrit cette requête ne connaît pas l'identifiant de l'équipe d'Angers.

Correction :

```
SELECT nom, prenom
FROM  Joueuse
JOIN Equipe
ON Equipe.id_equipe = Joueuse.id_equipe
WHERE Equipe.nom = "Angers"
ORDER BY Joueuse.nom ASC ;
```

3. Les résultats des matchs sont aussi publiés sur le site web de la ligue. Par exemple, pour le match n° 10 qui a opposé l'équipe de Villeneuve d'Ascq à l'équipe de Bourges le 23/10/2021 on retrouve les informations suivantes :

	Match n° 10 **23/10/2021**	
Villeneuve d'Ascq	**73 | 78**	**Bourges**

Le score final du match a été de 73 points pour l'équipe de Villeneuve d'Ascq qui a joué à domicile (nom affiché à gauche sur la page) contre 78 points pour l'équipe de Bourges qui a joué en déplacement (nom affiché à droite sur la page).

a) À partir de l'analyse de cet exemple, proposer un schéma relationnel pour la table `Match`. Si des clés étrangères sont définies, préciser quelles tables et quels attributs elles référencent.

On peut proposer le schéma relationnel suivant :

`Match`

id_match	INT
date	DATE
#id_equipe_domicile	INT
#id_equipe_deplacement	INT
score_domicile	INT
score_deplacement	INT

#id_equipe_domicile et #id_equipe_deplacement sont des clés étrangères qui font référence à la relation `Equipe`.

b) Écrire la requête SQL qui permet l'insertion dans la table `Match` de l'enregistrement correspondant à l'exemple donné ci-dessus.

Correction :

```
INSERT INTO Match
VALUES (10, "23/10/2021", 3, 6, 73, 78) ;
```

4. En plus du score final, sur la page web sont affichés des informations relatives aux performances des joueuses pendant le match.

Nous allons retenir ici seulement 3 critères : le nombre de points marqués, les rebonds et les passes décisives effectués.

Voici un extrait des statistiques du match n° 53 qui a opposé l'équipe de Landerneau à celle de Charleville-Mézières le 16/04/2022 :

Match n° 53
16/04/2022

Landerneau **56 | 64** **Charleville-Mézières**

Extrait statistiques :

Equipe	Nom	Prénom	Points	Rebonds	Passes décisives
Charleville-Mézières	Pouye	Tima	18	6	2
Charleville-Mézières	Akhator	Evelyn	15	17	0
Charleville-Mézières	Bouferra	Amel	10	3	9
Landerneau	Mane	Marie	18	2	3
Landerneau	Amukamara	Promise	12	2	5
Landerneau	Geiselsoder	Luisa	4	10	2

a) Proposer un schéma relationnel pour stocker les informations relatives aux statistiques des joueuses dans la base de données, telles que présentées ci-dessus.

On peut proposer le schéma relationnel suivant :

Statistiques

id_stats	INT
#id_joueuse	INT
#id_match	INT
points	INT
rebonds	INT
passes_decisives	INT

b) Écrire la requête SQL qui a été utilisée pour afficher la partie « Extrait des statistiques »de l'exemple ci-dessus.

Correction :

```
SELECT Equipe.nom, Joueuse.nom, Joueuse.prenom, Statistiques.points,
    Statistiques.rebonds, Statistiques.passes_decisives
FROM Statistiques
JOIN Joueuse ON Joueuse.id_joueuse = Statistiques.id_joueuse
JOIN Equipe ON Joueuse.id_equipe = Equipe.id_equipe
WHERE Statistiques.id_match = 53 ;
```

Exercice 2 — 3 points

Cet exercice porte sur la gestion des processus et la programmation orientée objet

On rappelle qu'un processus est l'instance d'un programme en cours d'exécution. Il est identifié par un numéro unique appelé PID. L'ordonnanceur est la composante du système d'exploitation qui gère l'allocation du processeur entre les différents processus. Nous allons nous intéresser à l'algorithme d'ordonnancement du tourniquet dont le fonctionnement est résumé ci-dessous :

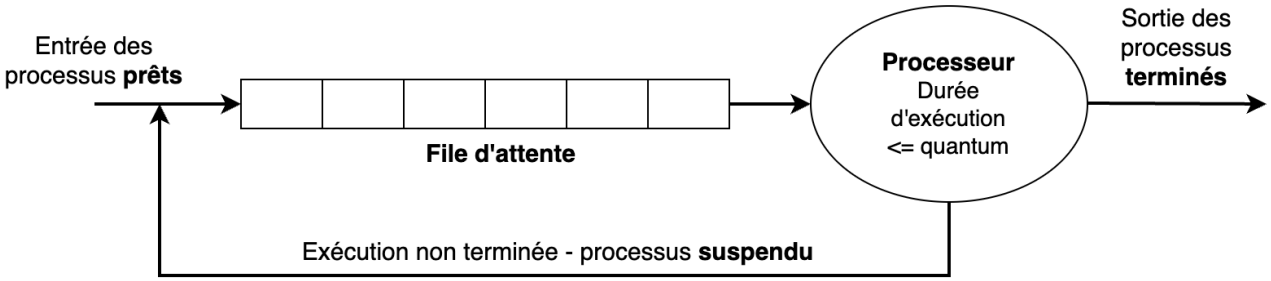

Schéma d'ordonnancement du tourniquet

- Les processus prêts à être exécutés sont placés dans une file d'attente selon leur ordre d'arrivée ;
- L'ordonnanceur alloue le processeur à chaque processus de la file d'attente un même nombre de cycles CPU, appelé **quantum** ;
- Si le processus n'est pas terminé au bout de ce temps, son exécution est suspendue et il est mis à la fin de la file d'attente ;
- Si le processus est terminé, il sort définitivement de la file d'attente.

1. On considère trois processus soumis à l'ordonnanceur **au même instant** pour lesquels on donne les informations ci-dessous :

PID	**Durée** (en cycles CPU)	**Ordre d'arrivée**
11	4	1
20	2	2
32	3	3

a) Si le quantum du tourniquet est d'un cycle CPU, recopier et compléter la suite des PID des processus dans l'ordre de leur exécution :

11, 20, 32, 11, 20, 32, 11, 32, 11

b) Donner la composition de la suite des PID lorsque le quantum du tourniquet est de deux cycles CPU.

Pour un quantum de 2 cycles CPU on obtient :

11, 11, 20, 20, 32, 32, 11, 11, 32

2. L'objectif de la suite de l'exercice est d'implémenter en langage Python l'algorithme du tourniquet.

Nous allons utiliser une liste pour simuler la file d'attente des processus et la classe `Processus` dont le constructeur est donné ci-dessous :

```
class Processus:
    def __init__(self, pid, duree):
        self.pid = pid
        self.duree = duree
        # Le nombre de cycle qui restent à faire :
```

```
        self.reste_a_faire = duree
        self.etat = "Prêt"
```

Les états possibles d'un processus sont : « *Prêt* », « *En cours d'exécution* », « *Suspendu* »et « *Terminé* ».

a) Recopier et compléter l'instruction Python suivante permettant de créer la liste d'attente initiale des processus donnés dans le tableau précédent (le processus PID 11 est à l'indice 0 de la liste d'attente) :

```
liste_attente = [Processus(...,...), ..........., .........]
```

Correction

```
liste_attente = [Processus(11, 4), Processus(20, 2), Processus(32,3)]
```

b) Recopier (sans les commentaires) et compléter les trois méthodes suivantes de la classe `Processus` :

```
    def execute_un_cycle(self):
        """Met à jour le reste à faire après l'exécution d'un
        cycle."""
        ..........................................

    def change_etat(self, nouvel_etat):
        """Change l'état du processus avec la valeur passée en
        paramètre."""
        ..........................................

    def est_termine(self):
        """Renvoie True si le processus est terminé, False sinon,
        en se basant sur le reste à faire."""
        ..........................................
```

Correction

```
def execute_un_cycle(self):
    """Met à jour le reste à faire après l'exécution d'un
    cycle.
    """
    self.reste_a_faire -= 1

def change_etat(self, nouvel_etat):
    """Change l'état du processus avec la valeur passée en
    paramètre.
    """
    self.etat = nouvel_etat

def est_termine(self):
    """Renvoie True si le processus est terminé, False sinon,
```

```
        en se basant sur le reste à faire."""
        return self.reste_a_faire <= 0:
```

c) La fonction `tourniquet` ci-dessous implémente l'algorithme décrit dans l'exercice.
Elle prend en paramètre une liste d'objets `Processus` donnés par ordre d'arrivée et un nombre entier positif correspondant au quantum. La fonction renvoie la liste des PID dans l'ordre de leur exécution par le processeur.
Recopier et compléter sur la copie le code manquant.

```
def tourniquet(liste_attente, quantum):
    ordre_execution = []
    while liste_attente != []:
        # On extrait le premier processus
        processus = liste_attente.pop(0)
        processus.change_etat("En cours d'exécution")
        compteur_tourniquet = 0
        while ..................... and .....................:
            ordre_execution.append(...........)
            processus.execute_un_cycle()
            compteur_tourniquet = compteur_tourniquet + 1
        if .....................:
            processus.change_etat("Suspendu")
            liste_attente.append(processus)
        else:
            processus.change_etat(...........)
    return ordre_execution
```

Correction

```
def tourniquet(liste_attente, quantum):
    ordre_execution = []
    while liste_attente != []:
        # On extrait le premier processus
        processus = liste_attente.pop(0)
        processus.change_etat("En cours d'exécution")
        compteur_tourniquet = 0
        # Tant que le processus n'est pas terminé
        # et que le quantum n'est pas atteint
        while not processus.est_termine() \
                and compteur_tourniquet < quantum:
            # On ajoute le processus à l'exécution
            ordre_execution.append(processus.pid)
            processus.execute_un_cycle()
            compteur_tourniquet = compteur_tourniquet + 1
        # Si le processus n'est pas terminé mais que
        # le quantum est terminé
        if not processus.est_termine() :
            processus.change_etat("Suspendu")
            liste_attente.append(processus)
        # Sinon, le processus est terminé
        else:
```

```
23                 processus.change_etat("Terminé")
24         return ordre_execution
```

Exercice 3 — 5 points

Cet exercice porte sur l'algorithmique, la programmation orientée objet et la méthode diviser-pour-régner

L'objectif de cet exercice est de trouver les deux points les plus proches dans un nuage de points pour lesquels on connaît les coordonnées dans un repère orthogonal.

On rappelle que la distance entre deux points A et B de coordonnées $(x_A\,;\,y_A)$ et $(x_B\,;\,y_B)$ est donnée par la formule : $AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$.

Les coordonnées d'un point seront stockées dans un tuple de deux nombres réels.

Le nuage de points sera représenté en Python par une liste de tuples de taille n, n étant le nombre total de points. On suppose qu'il n'y a pas de points confondus (mêmes abscisses et mêmes ordonnées) et qu'il y a au moins deux points dans le nuage.

Pour calculer la racine carrée, on utilisera la fonction `sqrt` du module `math`, pour rappel :

```
1 >>> from math import sqrt
2 >>> sqrt(16)
3 4.0
```

1. Cette partie comprend plusieurs questions générales :

a) Donner le rôle de l'instruction de la ligne 1 du code précédent.

La ligne 1 permet de réaliser l'import de la bibliothèque `math` afin de pouvoir utiliser la fonction `math`.

b) Expliquer le résultat suivant :

```
>>> 0.1 + 0.2 == 0.3
False
```

Cette instruction renvoie une `False` à cause des erreurs d'arrondis sur les nombres à virgule flottante. En effet, les nombres à virgule flottante sont représentés par une somme de puissance de 2. Comme 0,1, 0,2 et 0,3 ne peuvent pas s'exprimer comme une somme finie de puissance de 2, l'opération booléenne précédente renvoie `False`.

c) Expliquer l'erreur suivante :

```
>>> point_A = (3, 4)
>>> point_A[0]
3
>>> point_A[0] = 2
Traceback (most recent call last):
    File "<console>", line 1, in <module>
TypeError : 'tuple' object does not support item assignment
```

La variable `point_A` est un `tuple`. C'est une structure de donnée non mutable. Les valeurs qu'elle contient ne peuvent pas être modifiées.

2. On définit la classe `Segment` ci-dessous :

```
1 from math import sqrt
2 class Segment:
3     def __init__(self, point1, point2):
4         self.p1 =  point1
5         self.p2 = point2
6         self.longueur = ..............  # à compléter
```

a) Recopier et compléter la ligne 6 du constructeur de la classe `Segment`.

Correction :

```
6  self.longueur = sqrt((point2[0]-point1[0])**2 + (point2[1]-point1[1])**2)
```

La fonction `liste_segments` donnée ci-dessous prend en paramètre une liste de points et renvoie une liste contenant des objets `Segment` qu'il est possible de construire à partir de ces points. On considère **les segments [AB] et [BA] comme étant confondus et ajoutera un seul objet dans la liste**.

```
1 def liste_segments(liste_points):
2     n = len(liste_points)
3     segments = []
4     for i in range(.............):
5         for j in range(............., n):
6             # On construit le segment à partir des points i et j.
7             seg =  ..............................................
8             segments.append(seg)  # On l'ajoute à la liste
9     return segments
```

b) Recopier la fonction sans les commentaires et compléter le code manquant.

Correction :

```
1 def liste_segments(liste_points):
2     n = len(liste_points)
3     segments = []
4     for i in range(n-1):  # On s'arrête à l'avant dernier élément
5         for j in range(i+1, n):  # On commence à partir du i+1
6             # On construit le segment à partir des points i et j.
7             seg =  Segment(liste_points[i], liste_points[j])
8             segments.append(seg)  # On l'ajoute à la liste
9     return segments
```

c) Donner en fonction de n la longueur de la liste `segments`. Le résultat peut être laissé sous la forme d'une somme.
On cherche une combinaison de 2 éléments (2 points) parmi n éléments.

$$(n-1)+(n-2)+\ldots+1=\frac{n(n-1)}{2}$$

d) Donner, en fonction de n, la complexité en temps de la fonction `liste_segments`.
La complexité de cet algorithme est quadratique en $O(n^2)$ car il y a une boucle imbriquée.

3. L'objectif de cette partie est d'écrire la fonction de recherche des deux points les plus proches en utilisant la méthode diviser-pour-régner.

On dispose de deux fonctions : `moitie_gauche` (respectivement `moitie_droite`) qui prennent en paramètre une liste et qui renvoient chacune une nouvelle liste contenant la moitié gauche (respectivement la moitié droite) de la liste de départ. Si le nombre d'éléments de celle-ci est impair, l'élément du center se trouve dans la partie gauche.

Exemples :

```
>>> liste = [1, 2, 3, 4]
>>> moitie_gauche(liste)
[1, 2]
>>> moitie_droite(liste)
[3, 4]
```

```
>>> liste = [1, 2, 3, 4, 5]
>>> moitie_gauche(liste)
[1, 2, 3]
>>> moitie_droite(liste)
[4, 5]
```

a) Écrire la fonction `plus_court_segment` qui prend en paramètre une liste d'objets `Segment` et renvoie l'objet `Segment` dont la longueur est la plus petite.
On procédera de la façon suivante :

- Tester si le cas de base est atteint, c'est-à-dire lorsque la liste contient un seul segment ;
- Découper la liste en deux listes de tailles égales (à une unité près) ;
- Appeler récursivement la fonction pour rechercher le minimum dans chacune des deux listes ;
- Comparer les deux valeurs récupérées et renvoyer la plus petite des deux.

Correction :

```
1 def plus_court_segment(liste_segments):
2     """Renvoie un segment dont a longueur est la plus petite"""
3     if len(liste_segments) == 1:
4         return liste_segments[0]
5     else:
6         seg_gauche =  plus_court_segment(moitie_gauche(liste_segments))
7         seg_droite = plus_court_segment(moitie_droite(liste_segments))
8         if seg_gauche.longueur >= seg_droite.longueur:
```

```
            return seg_droite
        else:
            return seg_gauche
```

4. On considère les trois points A(3 ; 4), B(2 ; 3) et C(−3 ; −1).

a) Donner l'instruction Python permettant de construire la variable `nuage_points` contenant les trois points A, B et C.

Correction :

```
nuage_points = liste_segments([(3, 4), (2, 3), (-3, -1)])
```

b) En utilisant les fonctions de l'exercice, écrire les instructions Python qui affichent les coordonnées des deux points les plus proches du nuage de points `nuage_points`.

Correction :

```
>>> segment = plus_court_segment(nuage_points)
>>> print(segment.p1)
>>> print(segment.p2)
```