Amérique du Nord – 2026 – sujet2

Exercice 3 (8 points)

Cet exercice porte sur les bases de données relationnelles, le langage SQL et la programmation Python, en particulier les dictionnaires.

Un club d'athlétisme organise une compétition de course à pied sur route. Deux épreuves sont proposées : une course de 5 km et une course de 10 km.

Partie A : Bases de données

Dans cet exercice, 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`), `JOIN ... ON` ;
- construire des requêtes d'insertion et de mise à jour à l'aide de `UPDATE`, `INSERT` et `DELETE` ;
- affiner les recherches à l'aide de `DISTINCT` et `ORDER BY` ; réaliser des agrégations à l'aide de `COUNT`.

Le club souhaite gérer les inscriptions et les résultats de cette compétition à l'aide d'une base de données informatique constituée de deux tables : `coureur` et `epreuve`. Le schéma relationnel de cette base de données est représenté cidessous. Sur ce schéma, les clés primaires de chacune des tables sont soulignées et les clés étrangères sont précédées du symbole #.

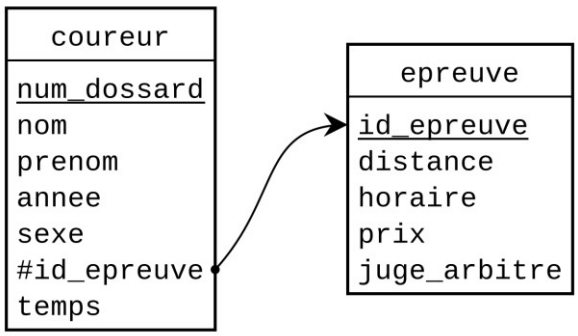

Figure 1. Schéma de la base de données

Chaque coureur ne peut participer qu'à une seule épreuve.

Relation `coureur`

L'identifiant du coureur est son numéro de dossard. Ce numéro est automatiquement incrémenté d'une unité à chaque nouvel enregistrement. Les autres champs doivent être saisis par l'organisateur lorsqu'il reçoit les bulletins d'inscription des coureurs. La codification du sexe est « H » pour les hommes et « F » pour les femmes. Les temps sont exprimés en secondes et sont initialisés avec la valeur 0.

Extrait de table coureur						
num_dossard	nom	prenom	annee	sexe	id_epreuve	temps
1	DA SILVA	José	1980	H	1	0
2	HANG LI	Léo	2005	H	2	0
3	BODIANE	Lola	2000	F	2	0
4	BRELET	Sandra	1972	F	1	0

Relation **epreuve**

Cette relation contient actuellement deux enregistrements, l'organisateur prévoyant d'ajouter de nouvelles distances dans le futur. La distance est exprimée en km et le prix de l'inscription en euros. Le champ `horaire` donne l'heure de départ de la course.

epreuve				
id_epreuve	distance	horaire	prix	juge_arbitre
1	5	10	10	GAVEAU Philippe
2	10	11	20	BOULA Awa

1. Expliquer le choix de `num_dossard` comme clé primaire pour la relation `coureur`.

2. Décrire ce que donne la requête SQL suivante :

```
SELECT nom, prenom
FROM coureur
ORDER BY nom ;
```

3. Écrire une requête SQL permettant d'établir le nom et prénom de toutes les femmes inscrites à la compétition.

4. Écrire une requête SQL permettant de connaître le nombre total d'inscrits à la compétition.

L'organisateur reçoit le bulletin d'inscription contenant les informations suivantes :

Nom : REMY ; Prénom : Patrice
Civilité : homme
Course : 5 km
Année de naissance : 1973

5. Écrire une requête SQL permettant d'insérer ce participant dans la base.

Le coureur dont le numéro de dossard est le 137 s'est blessé quelques jours avant l'épreuve, il ne pourra pas participer à la course.

6. Écrire une requête SQL qui permettra à l'organisateur de supprimer son inscription de la base de données.

Le coureur dont le numéro de dossard est le 256 a oublié sur quelle épreuve (5 km ou 10 km) il s'est inscrit. Il contacte l'organisateur pour qu'il lui rappelle la distance qu'il devra parcourir ainsi que l'horaire de son départ.

7. Écrire une requête SQL permettant de lui fournir ces informations.

Lorsque les coureurs franchissent la ligne d'arrivée, le juge arbitre note leur numéro de dossard et leur temps de course (en secondes) sur une feuille de pointage. Voici un extrait d'une telle feuille :

dossard	temps
57	1242
72	1845
183	1284
2	1285

Les temps de tous les coureurs ayant été enregistrés, on souhaite afficher le classement général de la catégorie *Master Femme* pour la course de 10 km. Cette catégorie regroupe les coureuses nées avant 1986.

8. Écrire une requête SQL renvoyant le numéro de dossard, le nom, le prénom et le temps de course de ces participantes classées par temps de course croissant.

PARTIE B : PROGRAMMATION PYTHON

Émilie, fille de l'organisateur et élève en classe de terminale spécialité NSI propose de réaliser une étude pluriannuelle des performances accomplies.

Pour enregistrer les informations elle crée pour chaque type de course un *dictionnaire de performances*. Les clés d'un tel dictionnaire sont les années où ce type de course a eu lieu. La valeur associée à une année est la liste des meilleurs temps réalisés dans chacune des six catégories suivantes (dans cet ordre).

- Junior homme (JH) et Junior femme (JF) : moins de 18 ans ;
- Sénior homme (SH) et Sénior femme (SF) : de 18 à 40 ans ;
- Master homme (MH) et Master femme (MF) : plus de 40 ans ;

Par exemple, le dictionnaire ci-dessous enregistre les performances pour les courses de 5 km. Il indique par exemple qu'en 2024 le meilleur temps réalisé dans la catégorie junior femme (JF) est de 1010 s, et celui dans la catégorie master homme (MH) de 1022 s.

```
dict_perf_5km = {2022 : [1020,1050, 900,1000,1018,1040],
                 2023 : [1010,1048,1100,1024,1080,1108],
                 2024: [1012,1010,1000,1036,1022,1098],
                 2025: [ 998,1028,1000, 959,1002, 980]}
```

9. Donner la valeur de l'expression `dict_perf_5km[2025][2]`

En 2026, pour la course de 5km, les meilleurs temps réalisés dans chaque catégorie sont les suivants :

Résultats par catégories - 2026					
JH	JF	SH	SF	MH	MF
1004	1016	1000	1140	1023	1024

10. Écrire l'instruction permettant d'ajouter ces informations dans le dictionnaire `dict_perf_5km`.

11. Écrire le code de la fonction `scratch` qui prend en paramètres un dictionnaire de performances `dico` et une année `annee`, et qui renvoie le meilleur temps (toutes catégories confondues) de cette année. On suppose que `annee` est une clé présente dans `dico`.
On n'utilisera pas les fonctions natives `min` et `max` de Python.

Par exemple, l'appel `scratch(dict_perf_5km, 2024)` renvoie `1000`.

Émilie propose maintenant la fonction suivante :

```
1  def mystere(dico, cat):
2      categories = ['JH','JF','SH','SF','MH','MF']
3      s = 0
4      nb = 0
5      for i in range(len(categories)):
6          if cat == categories[i]:
7              i_cat = i
8      for annee in dico.keys():
9          s = s + dico[annee][i_cat]
10         nb = nb + 1
11     return s/nb
```

12. Indiquer la valeur affectée à la variable `i_cat` au cours de l'appel `mystere(dict_perf_5km, 'SH')`.

Un appel du type `mystere(dict_perf_5km,'SG')` pose problème car `'SG'` n'est pas une catégorie répertoriée.

13. Indiquer quel sera le message d'erreur renvoyé à la console parmi les choix suivants :

- "`expected an indented block`";
- "`takes 2 positionnals arguments but 3 were given`";
- "`local variable 'i_cat' referenced before assignement`";
- "`list index out of range`".

14. Proposer une assertion à ajouter entre la ligne 2 et la ligne 3 pour indiquer une éventuelle faute de saisie par un message à l'utilisateur du script.

15. Indiquer le résultat de l'appel `mystere(dict_perf_5km,'SH')`.

Émilie souhaiterait disposer d'une fonction `records` prenant en paramètre un dictionnaire de performances et renvoyant la liste des meilleurs temps enregistrés pour chaque catégorie et toutes années confondues. Par exemple, l'appel `records(dict_perf_5km)` devrait renvoyer `[998, 1010, 900, 959, 1002, 980]`. On suppose que les temps enregistrés pour les différentes catégories et les différentes années n'excèdent jamais 24 h.

16. Écrire le code de la fonction `records`.