DS 1

Informatique de tronc commun, classe de PC

Julien Reichert

Ce devoir consiste en une partie de programmation et deux parties sur des bases de données totalement indépendantes. Les requêtes à écrire ne sont pas forcément de difficulté croissante.

Partie 1 : Programmation

On recommande sans l'imposer l'utilisation d'une structure de dictionnaire pour tous ces exercices de programmation. La complexité n'est ni à optimiser ni à calculer.

Exercice 1.1 : Écrire une fonction qui prend en argument une liste et qui détermine un élément apparaissant le plus souvent.

Exercice 1.2 : Écrire une fonction qui prend en argument deux listes et qui détermine si elles ont les mêmes éléments peu importe le nombre d'occurrences. Si des éléments sont égaux sans être les mêmes, comme par exemple `1` et `1.0`, on peut admettre que c'est bon malgré tout.

Exercice 1.3 : Même exercice mais le nombre d'occurrences doit aussi être le même.

Exercice 1.4 : On cherche si une liste d'entiers peut laisser apparaître des fréquences d'apparition croissantes puis décroissantes en considérant les valeurs elles-mêmes. Autrement dit, les valeurs sont certes désordonnées, mais la plus petite doit apparaître moins souvent que la deuxième plus petite, et ainsi de suite, jusqu'à un moment où les valeurs commencent à apparaître moins fréquemment, sans tendance à la hausse par la suite. Écrire une fonction qui prend en argument une liste et qui détermine si la condition attendue est respectée.

Par exemple, pour la liste `[44, 42, 41, 43, 40, 43, 41, 42, 42, 43]`, le nombre d'apparitions des valeurs est une pour `40`, deux pour `41`, trois pour `42` et `43`, ce qui finit la phase croissante puisque cela baisse à une apparition pour `44`. La réponse attendue est `True`, mais en ajoutant où que ce soit deux occurrences de `45`, la réponse deviendrait `False`.

Partie 2 : Base de données « football »

Voici la base de données pour la deuxième partie. Il s'agit de gérer des paris amicaux sur des matchs de la coupe du monde, avec un calcul de score fait par ailleurs (Python / PHP / etc.) en général. Pour chaque match, annoncer le bon résultat (équipe 1 gagne / match nul / équipe 2 gagne) donne un point en vue d'une première fonction de score, et annoncer le bon score donne un point en vue d'une deuxième fonction de score. On considère toujours le score après le temps règlementaire sans tenir compte d'éventuelles prolongations voire de tirs au but, du moins dans la base de données.

La base de donnée est constituée de cinq tables :

- `Equipes`, dont les attributs sont `Id_equipe` (entier) et `Pays` (chaîne de caractères) ;
- `Matchs`, dont les attributs sont `Id_match` (entier), `Date` (chaîne de caractères au format `"MM/JJ"`), `Phase` (chaîne de caractères pouvant valoir `"Groupe A"`, `"Quart de finale"`, ...), `Equipe1` (entier, l'équipe dite « receveuse ») et `Equipe2` (entier, l'équipe dite « visiteuse ») ;
- `Parieurs`, dont les attributs sont `Id_parieur` (entier) et `Nom_parieur` (chaîne de caractères) ;
- `Paris`, dont les attributs sont `Parieur` (entier), `Id_match` (entier), `Score1` (entier) et `Score2` (entier).
- `Resultats`, dont les attributs sont `Id_match` (entier), `Buts1` (entier) et `Buts2` (entier).

Exemples d'enregistrements (avec les attributs dans l'ordre) pour chaque table avec explications si besoin :

- Table `Equipes` : `(1, "France"), mais aussi (19, "Australie")`. Plutôt intuitif. L'identifiant est arbitraire.
- Table `Matchs` : `(5, "11/22", "Groupe D", 1, 19)`. Allez les Bleus !
- Table `Parieurs` : `(1, "Julien")`. Intuitif aussi.
- Table `Paris` : `(1, 5, 3, 0)`. Je pense que la France va gagner contre l'Australie 3 à 0.
- Table `Resultats` : `(5, 2, 0)`. En étudiant le score que j'ai annoncé, on déduit que j'ai pronostiqué une victoire de la France. Comme le résultat est conforme, cela me fait un point pour le premier classement. Cependant, ce n'est pas le résultat exact du match donc je ne marque rien pour le deuxième classement.

Exercice 2.1 : Décrire pour chaque table des clés pertinentes, en mentionnant les clés étrangères.

Exercice 2.2 : Pourquoi les points marqués par les parieurs ne figurent-ils dans aucune table ?

Exercice 2.3 : Serait-il envisageable de rassembler des tables ? Discuter des avantages et des inconvénients.

Exercice 2.4 : Écrire une requête permettant de connaître le nom de l'équipe d'identifiant `19` (je sais, c'est l'Australie...) et une requête permettant de connaître l'identifiant du Qatar.

Exercice 2.5 : Écrire une requête permettant de connaître le nom des équipes jouant la finale.

Exercice 2.6 : Écrire une requête permettant de connaître le nom des équipes ayant un match le premier jour. On ne se servira pas du fait que ce match est unique et d'identifiant `1` (ni de la connaissance du jour de ce match).

Exercice 2.7 : Écrire une requête permettant de récupérer tous les paris ayant donné lieu à un point en vue du premier classement (match nul bien pronostiqué ou la bonne équipe a gagné). Si d'autres attributs sont également présents dans le cadre d'une jointure, ce n'est pas problématique.

Exercice 2.8 : Même question pour le deuxième classement.

Exercice 2.9 : En admettant l'accès à une table virtuelle `succes` reprenant avec les attributs utiles le résultat des requêtes des questions suivantes (dont on suppose qu'elles fonctionnent), écrire une requête qui extrait de cette table le nombre de points par parieur (on peut se contenter de l'identifiant du parieur qui devra figurer aussi dans le résultat) avec un tri décroissant par points.

Exercice 2.10 : Faire exactement le même travail que les exercices 7 à 9 en Python à partir de deux listes `PARIS` et `RESULTATS` dont les éléments respectifs sont des listes de quadruplets pour la première et de triplets pour la seconde, en reprenant l'ordre dans les tables éponymes.

Pour ce dernier exercice, il faudra donc une fonction qui retourne une liste extrayant les paris réussis (ou directement les points par joueur) pour chacun des deux systèmes de points, et une fonction qui produit le classement général à partir de n'importe laquelle de ces deux listes.

L'algorithme de tri est au choix mais à écrire soi-même pour avoir les points correspondant au tri. À défaut, le tri par des méthodes internes à Python est autorisé en vue de marquer les points correspondant à la récupération du classement (il faudra alors correctement utiliser la fonction de tri).

Partie 3 : Base de données « numismatique »

Avec cette base de données inédite en DS (et qui sera sans doute construite à terme), je compte gérer ma collection de pièces étrangères ou anciennes.

La structure est la suivante :

- Table `Pieces`, dont les attributs sont `id_piece` (entier), `pays` (chaîne de caractères), `valeur` (flottant), `unite` (entier), `annee` (entier) et `valable` (booléen).
- Table `Unites`, dont les attributs sont `id_unite` (entier), `symbole` (chaîne de caractères) et `nom_unite` (chaîne de caractères).
- Table `Cours`, dont les attributs sont `monnaie` (entier) et `euros` (flottant).
- Table `Rangement`, dont les attributs sont `piece` (entier), `valise` (entier), `compartiment` (entier), `rangee` (entier) et `colonne` (entier).

Quelques explications peuvent être nécessaires :

- Plusieurs pièces identiques peuvent être rangées, et potentiellement empilées au même endroit (donc dans le même compartiment de la même valise, à la même colonne de la même rangée), mais parfois à côté, en tout cas la flexibilité nécessaire fait qu'il n'y a pas d'attribut pour le nombre d'exemplaires (tant mieux en vue des exercices à venir).
- Certaines pièces n'ont plus cours légal, d'où le booléen `valable`. Pour simplifier, cela correspond exactement à une unité dont le cours n'est pas mentionné dans la table idoine. Pour les autres monnaies, l'attribut `euros` détermine combien d'euros vaut un de la monnaie en question. Il y a un enregistrement pour les euros aussi, afin d'éviter de faire échouer les requêtes associées.
- La table `Cours` n'est pas rassemblée avec la table `Unite` pour éviter d'utiliser la valeur `NULL`, qui est hors-programme. Bien évidemment, dans une vraie base de données je ne m'en priverai pas.
- Les mises à jour de la table `Cours` seraient à faire régulièrement, mais le but de la collection n'est pas vraiment de spéculer...

Les clés primaires sont les suivantes :

- Table `Pieces` : `id_piece`.
- Table `Unites` : `id_unite`.
- Table `Cours` : `monnaie`.
- Table `Rangement` : `piece`.

Les clés étrangères sont les suivantes :

- Table `Pieces` : `unite` vers la table `Unites`.
- Table `Unites` : aucune.
- Table `Cours` : `monnaie` vers la table `Unites`.
- Table `Rangement` : `piece` vers la table `Pieces`.

Exercice 3.1 : Les doublons sont peu nombreux, à l'heure actuelle, donc cette structure reste raisonnable. Si on voulait ajouter une description de l'avers et du revers de chaque pièce, les deux attributs supplémentaires dans la table `Piece` correspondraient à des textes plutôt longs et les écrire plusieurs fois alourdirait la table donc une mesure préventive s'imposerait. Proposer une structure différente en mentionnant toutes les nouvelles tables et tous les nouveaux attributs des tables existantes pour pouvoir décrire chaque côté de chaque pièce avec deux attributs distincts, de sorte que deux pièces ayant les mêmes attributs et la même description n'occasionnent qu'un enregistrement de chaque description. Plusieurs propositions pertinentes sont possibles en pratique. Cette nouvelle structure ne sera **pas** utilisée dans les exercices suivants.

Exercice 3.2 : Écrire une requête déterminant le nombre total de pièces dont je dispose.

Exercice 3.3 : Écrire une requête déterminant le nombre total de pièces distinctes, c'est-à-dire le nombre d'enregistrements uniques en excluant l'identifiant des pièces.

Exercice 3.4 : Écrire une requête déterminant le nombre de valises dont je dispose et ayant au moins une pièce.

Exercice 3.5 : Écrire une requête déterminant la valise contenant le plus de pièces.

Exercice 3.6 : Écrire une requête déterminant la valeur faciale, le nom de l'unité monétaire, l'endroit où est rangée et l'âge de la plus vieille pièce dont je dispose.

Exercice 3.7 : Écrire une requête déterminant le symbole de la monnaie la plus forte enregistrée dans la base de données.

Exercice 3.8 : Écrire une requête déterminant tous les endroits où sont rangées des pièces en złoty (il n'y a pas de piège sur la déclinaison en polonais dans cette question).

Exercice 3.9 : Écrire une requête déterminant la valeur d'échange de l'ensemble de ma collection, en excluant toutes les pièces n'ayant plus cours légal.

Exercice 3.10 : Reprendre l'exercice précédent en Python, à partir de deux listes `PIECES` et `COURS` dont les éléments respectifs sont des listes de sextuplets pour la première et de couples pour la seconde, en reprenant l'ordre dans les tables éponymes.