DS 0

Informatique pour tous, deuxième année

Julien Reichert

Durée : environ une heure et demie.

Le contexte de ce devoir est le comptage des points pour la compétition de programmation de cet été.

Rappel du score par exercice :

- 10 points par personne ayant résolu l'exercice en moins de 24 heures ;
- 10 points pour la première personne ayant résolu l'exercice (on admettra ici qu'il n'y a jamais d'égalité), sachant que cette personne soumet l'exercice suivant et que si cette personne n'existe pas l'exercice suivant est soumis par l'arbitre ;
- 10 points pour la personne ayant soumis l'exercice si personne ne l'a résolu dans un délai convenu ;
- 1 point pour la personne ayant soumis l'exercice par concurrent ne l'ayant pas résolu, à l'exception des concurrents n'ayant résolu aucun exercice ;
- Aucun point pour une éventuelle résolution d'un exercice après plus de 24 heures si au moins une autre personne l'a résolu.

On considère que les données concernant la compétition sont rassemblées dans une base de données dont la structure est la suivante :

- Table `Exercice`, avec comme attributs `Id_exercice`, clé primaire avec auto-incrémentation, `Titre`, clé de type `VARCHAR` ; `Lien`, clé de type `VARCHAR` ; `Date`, de type `INT` et exprimant le moment de la soumission en tant que "timestamp" ; `Delai`, de type `INT` et exprimant le délai convenu en heures.
- Table `Concurrent`, avec comme attributs `Id_concurrent`, clé primaire avec auto-incrémentation ; `Identité`, clé de type `VARCHAR` ; `Classe`, de type `VARCHAR`.
- Table `Resolution`, avec comme attributs `Exo`, clé étrangère vers la clé primaire de la table en question ; `Concur`, idem ; `Date_resolution`, de type `INT` et exprimant le moment de la résolution en tant que "timestamp".

Un bref rappel sur le timestamp : il s'agit du nombre de secondes écoulées depuis un point de départ arbitraire, en l'occurrence le premier janvier 1970 à minuit pour les systèmes Unix.

Pour la partie en Python du devoir, on suppose l'existence de trois variables globales `exercice`, `concurrent` et `resolution` qui contiennent les mêmes données que les tables correspondantes, en tant que listes de dictionnaires. On ne tiendra pas compte du fait que les listes sont a priori triées.

Une brève introduction aux dictionnaires : il s'agit d'une structure de données pouvant être vue comme une liste indexée non pas par les nombres de zéro à la taille moins un mais par n'importe quelle valeur immutable appelée clé, les éléments stockés aux différentes clés étant les valeurs. On accède par exemple à la date de soumission du troisième exercice dans l'ordre de la liste par `exercice[2]["Date"]`. Un dictionnaire est mutable, avec la même syntaxe que pour les listes, et l'insertion d'une clé se fait en écrivant `dictionnaire[cle] = valeur`, écrasant l'éventuelle valeur déjà existante. La création d'un dictionnaire se fait par `dicovide = {}` ou `dicopasvide = {cle1 : valeur1, cle2 : valeur2}` en initialisant autant de valeurs qu'on veut. Pour mettre à jour une valeur numérique, on peut tout à fait écrire `dictionnaire[cle] += ajout`, mais cela déclenche une erreur si la clé n'est pas présente. La présence d'une clé peut se tester simplement (et de manière optimisée) par le test `cle in dictionnaire`. **En particulier, le parcours d'un dictionnaire se fait selon les clés.**

Pour obtenir la version triée d'une séquence, on peut utiliser la fonction `sorted`, qui renvoie toujours une liste. La séquence reste inchangée.

Exercice 1 : Écrire une requête qui permet d'obtenir les identités des participants ayant résolu au moins un exercice.

Exercice 1bis : Écrire une fonction en Python qui a le même effet. Puisque les variables annoncées sont globales, elles peuvent être des arguments ou non, au choix. Attention cependant à ne pas causer d'erreur de syntaxe...

Exercice 2 : Écrire une requête qui permet d'obtenir la liste des résolutions survenues après le délai de 24 heures.

Exercice 2bis : Écrire une fonction en Python qui a le même effet.

Exercice 3 : Écrire une requête qui permet d'obtenir, pour chaque classe, le nombre de fois où un étudiant de cette classe a été le premier à résoudre un exercice.

La requête de l'exercice 3 peut se décomposer ainsi :

- Récupérer tous les couples (Exercice, Date de résolution au plus tôt).
- Récupérer toutes les informations sur un étudiant pour lequel à un exercice donné la date de résolution coïncide avec cette date au plus tôt.
- Compter le nombre d'occurrences par classe.

Exercice 4 : Écrire une fonction qui prend en argument l'identifiant d'un exercice, qui a été posé par l'arbitre, et qui retourne un dictionnaire (ou une liste) permettant de déterminer le nombre de points marqués par chaque concurrent sur l'exercice.

Exercice 5 : Écrire une fonction qui prend en argument l'identifiant d'un exercice et celui de la personne qui l'a posté (et qui n'est pas l'arbitre) et qui retourne un dictionnaire (ou une liste) permettant de déterminer le nombre de points marqués par chaque concurrent sur l'exercice.

Exercice 6 : Écrire une fonction qui retourne un dictionnaire (ou une liste) permettant de déterminer le nombre de points marqués sur l'ensemble des exercices pour chaque concurrent. On rappelle que le premier exercice a été posé par l'arbitre et on signale que cette personne, ne pouvant pas marquer de points, n'est pas dans la base de données.