Donnez les réponses sur le sujet que vous joindrez à votre copie.

Strava : contraintes et requêtes

Dans tous les exercices suivants, on considère une simplification de la base de données utilisée par l'application running, vélo et randonnée Strava afin d'enregistrer et comparer les performances sportives des athlètes inscrits. La base de données respecte le schéma ci-dessous, où les clefs primaires sont soulignées et les clefs étrangères listées à la suite.

`athlete(idAt, pseudo, mdp, email, age, gender, weight, city, country)`
`activity(idA, idAt, type, description, startDate, distance, movingTime, elapsedTime)`
`segment(idS, type, description, latStart, latEnd, longStart, longEnd, distance)`
`segmentEffort(idA,idS, startDate, endDate, elapsedTime)`

Clefs étrangères : `idAt` dans `activity` fait référence à `idAt` dans `athlete` ; `idA` dans `segmentEffort` fait référence à `idA` dans `activity` et `idS` dans `segmentEffort` fait référence à `idS` dans `segment`.

Les données concernant les athlètes sont stockées dans la table `athlete` (`mdp` correspond au mot de passe du compte de l'athlète). Les activités des athlètes peuvent être de plusieurs types (par exemple, `ride` pour le cyclisme, `run` pour la course à pied). Elles sont associées à une description faite par l'athlète de sa propre activité, par exemple `Longchamp chill 100 bornes à 39km/h`. Les activités peuvent comporter des correspondances avec un ou plusieurs segments du même type, i.e., une activité est associée à un segment si celui-ci est entièrement contenu dans l'activité. Les segments sont également associés à une description ; par exemple, `Pont neuf sprint` pour le segment de type `ride` qui traverse à Paris le Pont Neuf du Nord au Sud. Ainsi, un cycliste qui traverse le Pont Neuf du Nord au Sud au cours d'une activité obtiendra une performance sur ce segment, laquelle sera stockée dans la table `segmentEffort`. Seul le temps écoulé est stocké pour les performances de segment, alors que pour les activités, on distingue entre le temps écoulé (`elapsedTime`) et le temps en mouvement (`movingTime`, tous deux stockés en secondes), les compteurs pouvant être mis en pause au cours d'une activité. Seuls les coordonnées de départ et d'arrivée des segments (latitude `latStart, latEnd` et longitude `longStart, longEnd`) sont stockées dans les données relationnelles. L'exacte correspondance est faite au niveau des fichiers au format `gpx` récupérant les activités des athlètes, mais nous ferons abstraction ici de cet aspect. Ainsi, il est tout à fait possible que deux segments distincts aient les mêmes point de départ et d'arrivée, mais pas la même longueur (attribut `distance` stocké en mètres) s'ils correspondent à des chemins passant par des points intermédiaires différents entre le point de départ et le point d'arrivée. Une des fonctionnalité les plus populaires de Strava est la comparaison des performances des athlètes sur les différents segments, l'athlète ayant obtenu le meilleur temps devenant `KOM` (King of the Mountain) pour les hommes et `QOM` (Queen of the Mountain) pour les femmes sur ce segment ou bien `CR` (Course Record) au classement général (quel que soit le genre). Attention, plusieurs athlètes peuvent partager une même couronne s'ils ont tous réalisé le temps minimal pour un certain segment (il y a donc des ex-aequos). Les dix meilleures performances de segment sont listées dans le `Top 10` (il y en a 3 distinctes : le classement général, le classement hommes et le classement femme).

Exercice 1 : schéma relationnel

Pour chacune des questions suivantes, écrivez `interdit` si le schéma ne permet pas les situations décrites, sinon écrivez `autorisé`. Si votre réponse vous permet de déceler quelque

chose de problématique, expliquez quels choix faire (contraintes, types de données, etc.) à la création des tables pour corriger le schéma.

La syntaxe SQL précise n'est pas obligatoire : indiquez seulement la nature du choix, les tables et attributs concernés, ainsi que les mots clefs nécessaires.

1) L'identifiant d'un athlète dans un tuple de la table `segmentEffort` n'est pas renseigné.

> **Solution:** Interdit. Cet attribut fait partie de la clef primaire.

2) Un athlète peut réaliser plusieurs performances au cours d'une même activité sur le même segment. Par exemple, un cycliste peut tourner plusieurs fois autour de l'anneau cyclable de Longchamp et donc obtenir des temps différents au cours de chaque activité pour chaque performance sur le segment `Longchamp` correspondant à un tour complet.

> **Solution:** Interdit : `startDate` ne fait pas partie de la clef primaire.

3) La valeur de l'attribut `endDate` est inférieure à celle de l'attribut `startDate` dans un tuple de la table `segmentEffort`.

> **Solution:** Autorisé. Ajouter une contrainte vérification afin d'imposer l'inégalité (mot clef `CHECK`).

4) Un segment sur lequel des athlètes ont enregistré des performances peut être retiré de la table `segment`.

> **Solution:** Interdit si rien n'a été prévu en amont lors de la création de la contrainte de clef étrangère. Peut être autorisé avec `ON DELETE CASCADE`.

5) La table `activity` contient des tuples dans lesquels la valeur de l'attribut `type` ne correspond à aucune valeur de l'attribut `type` dans les tuple de la table `segment`.

> **Solution:** Autorisé,mais il est facile de contraindre les valeurs possibles au moyen de contraintes de vérification (mot clef `CHECK`).

Exercice 2 : requêtes SQL

1) Ecrivez une vue `segEffortF(idAt,pseudo,idA,idS, startDate, endDate, elapsedTime)` permettant de restreindre les informations de performance de segments de type `ride` aux athlètes ayant sélectionné comme genre `'femme'`. Notez que la vue comportera également un attribut `idAt` représentant l'identifiant de l'athlète concernée. (Si vous ne savez pas comment écrire une vue, écrivez simplement la requête retournant ces résultats.)

> **Solution:**
>
> ```
> CREATE VIEW segEffortF AS
> (At.idAt, At.pseudo, SE.idA,SE.idS, SE.startDate, SE.endDate, SE.elapsedTime
> FROM segment S, segmentEffort SE, athlete At, activity A
> WHERE S.idS=SE.idS AND SE.idA=A.idA AND A.idAt=At.idAt
> AND At.gender='femme' AND S.type='ride' );
> ```

On suppose maintenant qu'on a à notre disposition la vue `segEffortF` créée à l'étape précédente, mais également une vue `segEffortH` restreignant de manière similaire les performances de segment aux athlètes ayant sélectionné comme genre `'homme'`. Proposez des requêtes SQL permettant d'extraire les informations demandées.

2) La liste des segments de type `ride` pour lesquels aucune femme n'a obtenu de performance de segment (tableau résultat : `idS, description`).

Solution:

```
(SELECT idS, description
FROM segment S
WHERE type='ride')
EXCEPT
(SELECT idS, description
FROM segmentEffortF) ;
```

3) La liste des segments de type `ride` pour lesquels la QOM a obtenu un meilleur temps (i.e., plus court) que le KOM (tableau résultat : `idS, description`).

Solution:

```
SELECT idS, description
FROM segEffortF F
WHERE elapsedTime < ALL
(SELECT elapsedTime
FROM segEffortH H
WHERE F.idS=H.idS);
```

4) Les athlètes ayant une activité de chacun des types d'activité représenté dans la table activité (tableau résultat : (`idAt,psseudo`)).

Solution:

```
SELECT idAt, pseudo
FROM athlete NATURAL JOIN activity
GROUP BY idAt HAVING COUNT(DISTINCT type)=SELECT COUNT(DISTINCT type) FROM activity;
```

Cette version là est correcte également, bien que moins efficace sur de grosses données (chaque niveau d'imbrication de sous requête nécessite une coûteuse opération de jointure) :

```
SELECT idAt, pseudo
FROM athlete At
WHERE NOT EXISTS (SELECT * FROM activity
WHERE type NOT IN (SELECT A.type FROM activity A WHERE A.idAt=At.idAt);
```

5) Les paires d'athlètes femmes ayant au moins une performance de segment de type `ride` sur le même segment (tableau résultat : `idAt1, idAt2`). Veillez à ne pas retourner de doublons et à ne pas retourner le couple `(b, a)` si vous avez déjà retourné le couple `(a, b)`. Ne retournez pas non plus les couples triviaux de la forme `(a, a)`.

Solution:

```
SELECT DISTINCT S1.idAt as idAt1, S2.idAt as idAt2
FROM SegEffortF S1, SegEffortF S2
WHERE idAt1 < idAt2
AND S1.idS=S2.idS ;
```

6) La ou les athlète(s) ayant le plus de QOM sur des performances de type `ride` (tableau résultat : (`idAt`).

Solution:

```
WITH best AS (SELECT idS, MIN(elapsedTime) as time
               FROM SegEffortF
               GROUP BY idS)
      qom AS (SELECT idAt,pseudo, time
               FROM  SegEffortFemme S, best B
               WHERE S.idS=B.idS
               AND elapsedTime=time)
     nbQom AS(SELECT idAt,pseudo,COUNT(time) as nb
               FROM qom)
SELECT idAt,pseudo FROM nbQom WHERE nb = (SELECT max(nb) FROM nbQom);
```

(En toute rigueur il ne faudrai compter chaque QOM qu'une seule fois par utilisateur (ici on peut compter plusieurs performances de segments différentes pour un même athlète et un même segment si elles ont le même temps).

Exercice 3 : requêtes récursives avec postgres

1) On se restreint aux segments de type `ride` et on s'intéresse aux chemins passant entre le segment `Le meilleur segment de Paname` et le segment `Grands Moulins to Av de France`. On cherche à déterminer s'il existe une suite de segments les reliant et qui soit telle que les coordonnées de latitude et longitude de la fin de chacun des segments sur la suite soient égaux aux coordonnées de latitude et de longitude du début du prochain. (Ce n'est évidement pas très réaliste, car il vaudrait mieux considérer des coordonnées "suffisamment" proches, mais nous ferons abstraction de ces contraintes réalistes ici.) Votre requête retournera un résultat vide s'il n'existe pas de chemin de ce type, ou bien la description `Grands Moulins to Av de France` si un tel chemin existe.

Solution:

```
WITH RECURSIVE chain (description, latStart, latEnd, longStart, lon-
gEnd) AS (
SELECT type description, latStart, latEnd, longStart, longEnd
FROM segment
WHERE description = Le meilleur 'segment de Paname' AND type = 'ride'
UNION
SELECT S.type, S.description, S.latStart, S.latEnd, S.longStart, S.longEnd
FROM chain C, segment S
WHERE C.latEnd=S.latStart AND C.longEnd=S.longStart AND type='ride'
)
SELECT description FROM chain WHERE description='Grands Moulins to Av de France';
```

Exercice 4 : requêtes en algèbre relationnelle

On considère ici aussi que le schéma est enrichi par les vues `segmentEffortF` et `segmentEffortH` (vous pouvez donc les utiliser comme vous utiliseriez des tables. Proposez des requêtes d'algèbre relationnelle permettant d'extraire les informations demandées.

1) La liste des segments de type `ride` pour lesquels aucune femme n'a obtenu de performance de segment (tableau résultat : `idS, description`).

Solution:

2) Les paires d'athlète femmes ayant au moins une performance de segment sur le même segment (tableau résultat : `idAt1, idAt2`). Veillez à ne pas retourner de doublons et à ne pas retourner le couple `(b, a)` si vous avez déjà retourné le couple `(a, b)`. Ne retournez pas non plus les couples triviaux de la forme `(a, a)`.

Solution:

1 Exercice 5 : modélisation, reverse engineering, enrichissement et amélioration du schéma

1. Proposez une modélisation E/R correspondant au schéma relationnel décrit précédemment. Certains points sont problématiques, améliorez-les. Veillez par exemple à ajouter des contraintes de vérification, unicité ou non nullité quand c'est pertinent. On précisera les entités, les identifiants, les associations, on indiquera pour chaque association les cardinalités. Listez les contraintes externes avec soin et expliquez comment les implémenter lorsque ceci est possible avec les notions vues en cours.
2. On décide de raffiner notre modélisation. Certaines performances de segment sont parfois problématiques (par exemple, un KOM de type `ride` à 350km/h est hautement suspect. Il est probablement lié à une erreur de gps ou à une utilisation d'un compteur vélo dans un véhicule motorisé. On souhaite autoriser les utilisateurs de Strava à signaler des performances de segment de ce type. En revanche, pour éviter les abus, on bloquera temporairement la fonctionnalité de signalement si un utilisateur en abuse (on considérera qu'au delà de 20 signalements dans une même journée, le signalement est temporairement impossible pour les prochaines 24 heures). On souhaite aussi permettre aux athlètes de se suivre entre eux (attention, le suivi n'est pas forcément mutuel) et de s'envoyer des 'kudos' (sur le modèle des 'like' de facebook) et des commentaires pour les activités réalisées. Attention, un athlète ne peut pas s'envoyer de kudo à lui même et plusieurs commentaires peuvent être envoyés par un même athlète sur une même activité (y compris s'il s'agit de sa propre activité). Enfin, on souhaite représenter aussi le fait que des clubs relevant de différents types d'activité (e.g., pour le cyclisme, Velo Club de Neuilly, Ride Alternatif, Mayo Jaune, Saru ride, etc) peuvent avoir un compte sur `Strava`. Les athlètes peuvent s'inscrire à ces clubs et aux sorties qu'ils proposent (en revanche, les athlètes ne peuvent pas proposer de sorties). Les sorties sont définies par une description, une date et un point de rendez-vous. Attention, un club ne peut pas enregistrer d'activité et ne peut pas s'inscrire à une sortie (il peut simplement publier des sorties et avoir une liste d'athlètes inscrits sur lesquels des classements de performance peuvent être réalisés). Vous pourrez évidement être amené à modifier votre modélisation initiale de manière substantielle.

   Modifiez le schéma E/R que vous avez proposé afin d'accommoder ces nouveaux besoins. N'oubliez pas de lister les contraintes externes (il est tout à fait possible qu'une bonne partie du cahier des charges relève des contraintes externes !).
3. Proposez une traduction de votre diagramme dans le modèle relationnel. Précisez les clefs primaires et les contraintes référentielles (en particulier les clefs étrangères). Si certaines contraintes du modèle n'ont pas pu être implementées, précisez-le.