Projet de Base de Données

Réalisation d'une base de données de gestion de stations de ski

D. Pellier

Un ensemble de stations de ski vous a confié la conception d'une base de données pour la gestion des forfaits. A partir d'un entretien avec les responsables du regroupement, vous avez rédigé la description suivante.

Une station vend plusieurs types de forfaits qui ont chacun un identifiant, une description, un prix, une plage de validité dans la journée (par défaut de 9h à 17h), une durée de validité en jours, et éventuellement la description textuelle des conditions d'accès (étudiant, plus de 60 ans, moins de 12 ans, etc.).

Des skieurs achètent des forfaits (identifiés par un numéro) qui débutent soit à la date d'achat (si celle-ci est renseignée lors de la création du forfait) ou soit lors du premier passage à une remontée mécanique. Chaque forfait est associé à une carte magnétique qui permet de passer aux remontées mécaniques. Une carte magnétique est identifiée par un numéro. Dans des soucis de développement durable, les cartes peuvent être réutilisées (par l'achat d'un nouveau forfait) lorsque le forfait n'est plus valable. Une carte ne peut pas être associée à deux forfaits valides en même temps (ou non utilisés).

Grâce à son forfait, un skieur peut emprunter les remontées mécaniques de la station pendant la validité de son forfait. Cependant, pour éviter une utilisation frauduleuse, et pour maintenir des statistiques d'utilisation des remontées, tous les passages (heure+date de passage) d'un skieur dans une remontée mécanique sont enregistrés. Lorsqu'il emprunte une remontée son forfait est invalide (temporisé) durant le temps de cette remontée.

Les remontées sont identifiées par un numéro et décrites par leur nom, leur durée (durée passée sur cette remontée) et associées à un type de remontée (télésiège, téléski, télécabine, téléphérique, etc.) lui aussi identifié par un numéro.

Pour vous aider, nous vous donnons ci-dessous la relation non normalisé :

STATION(id_type_forfait, libelle_type_forfait, prix, heure_debut, heure_fin, duree_forfait, condition, id_carte, id_personne, date_debut, id_type_remontee, libelle_type_remontee, id_remontee, nom_remontee, duree_remontee, capacite id_forfait, heure_passage)

1 Consignes générales

- Le projet sera à rendre lors de la dernière séance de projet (cf. page du cours) (tout projet rendu en retard ne sera pas pris en compte).
- Vous pouvez réaliser le travail par binôme.
- Chaque binôme presentera son projet à l'enseignant lors de la dernière séance de cours pendanty 10 minutes.
- Le projet sera réalisé sur le SGBD PostgreSQL. De la documentation sur PostgreSQL est disponible à `http://docs.postgresqlfr.org/9.6/`
- Vous devrez installer le SGBD sur votre machine pour effectuer le projet. Vous trouverez les informations nécessaires à l'installation du SGBD à l'adresse `http://docs.postgresql.fr/9.6/INSTALL.html`

Vous devez rendre une archive ZIP contenant :
- le rapport (PDF) contenant la normalisation (question 1), ainsi que les tests des contraintes et de triggers que vous avez réalisés,
- un fichier texte (tables.sql) contenant les commandes de création des tables et de leurs contraintes,
- un fichier texte (triggers.sql) contenant les commandes de création des triggers,
- un fichier texte (alimentation.sql) contenant les requêtes qui permettent d'alimenter vos tables à partir de la table 'projet.bd_station',
- un fichier texte (requêtes.sql) contenant vos requêtes.

2 Conception et normalisation de la base de données

- Identifiez tous les champs que votre base de données va contenir (vous pouvez vous inspirer de la table 'projet.bd_ski').
- Identifiez les dépendances fonctionnelles.
- Quelle est la forme normale de la relation (avant décomposition) ?
- Proposez un schéma de base de données en 3ème Forme Normale.
- Identifiez les contraintes référentielles (clés étrangères) de votre schéma.
- Identifiez les contraintes de non nullité et de domaine que vous voulez vérifier sur les attributs.

3 Définition de la base de données

Ecrivez les créations de tables, ainsi que leurs contraintes (clés primaires, clés étrangères, non nullité, unicité).

Types de données à utiliser :
- chaîne de caractères : varchar(30) (tous les champs texte ont une longueur 30)
- entier : integer
- heure : time (format d'insertion 'hh :mm :ss')
- date+heure : timestamp (format d'insertion 'aa/mm/jj hh :mm :ss')
- durée en jours + heures : interval (format d'insertion 'jj hh :mm :ss')
- durée en jours : integer

4 Contraintes et déclencheurs

Certaines contraintes ne sont pas vérifiables directement :
- Une carte ne peut pas être associée à deux forfaits qui ont une période de validité commune.
- Une carte ne peut pas être associée à un nouveau forfait si l'ancien n'est pas encore utilisé (date_debut IS NULL)
- On ne peut passer à une remontée mécanique que si le forfait est valide
- On ne peut passer à une remontée mécanique que si le forfait n'est pas temporisé

Pour chacune de ces contraintes proposez une requête qui permet de vérifier le respect de la contrainte. Vous aurez sans doute besoin d'utiliser des fonctions qui permettent de manipuler des données temporelles. Vous trouverez toutes les informations utiles sur la page : `http://docs.postgresql.fr/9.6/functions-datetime.html`

Ecrivez ensuite les triggers correspondants. Les triggers ne seront exécutés que pour les insertions. Pour chaque trigger, proposez un exemple qui permet de vérifier son fonctionnement (une insertion qui fonctionne et une autre qui est rejetée par le trigger).

Note sur la réalisation de triggers : Chaque SGBD possède sa propre gestion des triggers (même si c'est un standard SQL maintenant). De plus, les triggers sont souvent écrits dans un langage procédural propre au SGBD. Dans le cas de PostgreSQL, nous utiliserons le langage procédural PL/PgSQL. La documentation sur l'utilisation de ce langage pour la réalisation de trigger est disponible sur : `http://docs.postgresql.fr/9.6/plpgsql-trigger.html`. De l'information plus générale sur les triggers : `http://docs.postgresql.fr/9.6/triggers.html`

5 Actions automatiques

Les triggers peuvent également servir à effectuer une procédure lorsqu'un événement (INSERT, UPDATE ou DELETE) arrive sur une table.

Dans le cadre de notre base de données de station de sports d'hiver, nous voulons réaliser l'action suivante : "si le forfait n'a pas de date de début renseignée alors cette date de début doit être renseignée lors du premier passage à une remontée mécanique". Ecrire le trigger qui met à jour la date de début d'un forfait (lorsque cette date est nulle) lors du premier passage. Proposez un exemple qui permet de vérifier son bon fonctionnement.

6 Requêtes

Pour répondre aux requêtes, vous alimenterez votre base, à partir de la relation (non normalisée) « projet.bd_station » que je vous fournis. Attention, cette table contient plus de 500 000 enregistrements. Les données qu'elle contient respectent toutes les contraintes (clés, unicité, triggers, etc.) introduites précédemment.

1. Quel est le dernier forfait valide correspondant à un identifiant de carte donné (exemple : carte n°1) ?
2. Quels sont les noms des remontées de type 'télésiège' ?
3. Quels sont les remontées de type 'télésiège' empruntées avec le forfait n°1 ?
4. Quelles sont les noms des remontées non empruntées avec le forfait n°2 ?
5. Pour chaque type de forfait, quel a été le nombre de forfaits vendus ?
6. Combien de forfaits ont été utilisés sur toutes les remontées de la station ?
7. Quelles sont les cartes qui ont été les plus réutilisées (c'est à dire associées au plus grand nombre de forfaits) ?
8. Quel est le nombre de passages enregistrés pour chaque remontée ?
9. Quel est, chaque jour, le nombre de passages enregistrés pour chaque remontée ?
10. Quelle est la remontée la plus fréquentée (où il y a eu le plus de passages) ?
11. Quel est le télésiège le moins fréquenté ?
12. Quel(s) forfait(s) a(ont) servi le plus de fois sur une journée ?
13. Quel est le chiffre d'affaire de la station (somme des prix des forfaits vendus) ?
14. Quel est le chiffre d'affaire de la station ventilé par mois (pour les forfaits à cheval sur 2 mois, on les compte par rapport à leur date de début de validité) ?

7 Divers

Si vous voulez installer PostgreSQL chez vous :
- sous Windows : `http://www.postgresql.org/download/windows`
- sous Linux UBUNTU : `http://doc.ubuntu-fr.org/postgresql`

Pour installer la base :
- décompressez le fichier dump_bd_station.sql.gz (gunzip dump_bd_station.sql.gz sous n'importe quel linux).
- créez une base de données : createdb nombase
- chargez les données dans votre base : psql -f dump_bd_station.sql nombase

8 Quelques compléments sur le projet

Les question 3 et 4 sont les plus difficiles. Faites la question 5 si vous n'y arrivez pas, puis revenez à ces deux questions à la fin.

8.1 Sur les champs

- heure_debut et heure_fin sont les heures de la journée entre lesquelles un forfait est valide. durée_forfait est le nombre de jours qu'un forfait est valide. Par exemples :
  - un forfait de type "journée" est valide de 9h à 17 pendant 1 journée
  - un forfait de type "semaine" est valide de 9h à 17h pendant 7jours
  - un forfait de type "matinée" est valide de 9h à 14h pendant 1 journée
- date_debut : c'est la date à laquelle un forfait commence. Cette date peut être renseigné (mais pas obligatoirement) lors de la vente du forfait. Si ce champ n'a pas été renseigné lors de la vente alors il le sera lors du premier passage à une remontée.

Ce forfait sera donc valide pendant les heures de validité de ce forfait entre date_debut et date_debut+duree_forfait.

8.2 Sur les triggers

Avant de faire cette question, il est souhaitable d'alimenter vos tables avec les données de la table "projet.bd_station" cf. question 5 sur les requêtes. Comme cette question est la plus délicate, vous pouvez la faire en dernier.

Rappel : Un trigger (ou déclencheur en français) est un petit programme qui est lancé automatiquement avant ou après une certaine action (insertion, suppression, mise à jour) sur une table donnée. Ces petits programmes servent par exemple à vérifier une contrainte non exprimable avec les contraintes classiques (NOT NULL, UNIQUE, PRIMARY KEY, REFERENCES, etc.). C'est ce que l'on veut faire dans cette question 3.

Les triggers que je vous demande d'écrire sont des triggers qui vérifient une contrainte lors de l'insertion. Ces triggers seront déclenchés avant la nouvelle insertion. Je vous explique ici comment faire les deux premiers triggers. Je vous laisse ensuite réfléchir sur les autres. Les 2 premières contraintes étant liées on va les traiter avec le même trigger. Ces contraintes sont à vérifier lorsque l'on veut ajouter un nouveau forfait.

Pour la contrainte 1, on va compter le nombre de forfaits qui ont une période de validité commune avec le nouveau forfait que l'on veut ajouter et qui partagent la même carte. Si ce

nombre n'est pas égal à 0 alors la contrainte n'est pas vérifiée. Pour la contrainte 2, on va compter le nombre de forfaits qui ont une date de début nulle et qui partagent la même carte. Si ce nombre n'est pas égal à 0 alors la contrainte n'est pas vérifiée. On peut facilement regrouper ces deux requêtes en 1 seule.

Tout d'abord on doit pouvoir comparer deux périodes de temps et savoir si elles se chevauchent. Dans postgres, il y a une fonction qui permet de vérifier si deux périodes se chevauchent : c'est la fonction OVERLAPS (cf. la doc de posgres). Sa syntaxe est la suivante :

```
( début1, fin1 ) OVERLAPS ( début2, fin2 )
( début1, longueur1 ) OVERLAPS ( début2, longueur2 )
```

Essayez les exemples suivants :

```
SELECT (DATE '2009-04-01', DATE '2009-04-01' + 1)
   OVERLAPS (DATE '2009-04-02',DATE '2009-04-03');
SELECT (DATE '2009-04-01', DATE '2009-04-01' + 7)
   OVERLAPS (DATE '2009-04-04',DATE '2009-04-04'+7);
```

Sur ces deux exemples, il faut rajouter DATE pour dire que la chaîne qui suit est de type DATE.

Comment faire la requête ?

Lorsque l'on veut ajouter un forfait, on connaît (ou l'on peut récupérer) : la date de début du forfait, la durée de validité du forfait et le numéro de la carte à laquelle il est affecté.

Si par exemple, l'id_carte est 123, la date de début est le '01-04-2009' et la durée de validité du forfait est 1 jour, alors la requête que vous devez faire ressemblera à :

```
SELECT count (id_forfait)
FROM ...
WHERE
(date_debut,date_debut+duree_forfait)
OVERLAPS (DATE '2009-04-01', DATE '2009-04-01' + 1) AND id_carte=123;
```

Je vous laisse rajouter la condition qui permet de vérifier en même temps si il n'y a pas de forfait ayant la même carte et une date de début égale à NULL.

Maintenant comment écrire le Trigger :

Sous postgresql, la création d'un trigger se déroule comme suit :

1. Création d'une fonction qui teste notre contrainte et qui lève une exception si la contrainte n'est pas respectée.
2. déclaration du trigger en utilisant la fonction précédemment créée.

8.2.1 La fonction

Lorsque l'on ajouter un nouveau forfait et la carte associée, on aura les cas suivants :

1. La date de début du nouveau forfait est renseignée
2. La date de début du nouveau forfait n'est pas renseignée

Dans le deuxième cas, on prendra alors la date du jour.

Dans le contexte d'un trigger (EACH ROW uniquement), la ligne qui va être ajoutée (supprimée, ou mise à jour) est stockée dans la variable NEW. Les valeurs des attributs de cette ligne sont accessibles par : NEW.date_debut, NEW.duree_forfait, et NEW.id_carte, etc.

Voici à quoi votre fonction doit ressembler.

```
CREATE OR REPLACE FUNCTION carte_utilisee() RETURNS trigger AS $carte_util$
  DECLARE
    -- commentaires commencent par deux tirets
    -- ici on declare les variables dont on a besoin
    -- si on n'a pas de variable à déclarer alors on ne met pas le mot DELCARE
    -- et commence directement par BEGIN

    -- ici je déclare une variable nb_forfaits de type INTEGER
    nb_forfaits INTEGER;
    new_date_debut DATE;
    new_duree_forfait INTEGER;
  BEGIN
    -- On teste la date de debut du nouveau forfait
    -- si elle est nulle alors on met la date du jour dans date_debut
    IF (NEW.date_debut IS NULL) THEN
        new_date_debut = (SELECT CURRENT_DATE);
    ELSE
        new_date_debut = NEW.date_debut;
    END IF;

    -- on récupère la durée du forfait
    new_duree_forfait = (SELECT duree_forfait FROM ... WHERE id_type_forfait=NEW.id_type_forfait);
    -- requete qui compte le nombre de forfait valides utilisant la carte
    nb_forfaits = (SELECT count (id_forfait) FROM ... WHERE (date_debut,date_debut+duree_forfait)
        OVERLAPS (new_date_debut, new_date_debut + new_duree_forfait) AND id_carte=NEW.id_carte);
    -- Ajoutez à la requête précédente la condition de vérification
    -- il n'existe pas de forfait avec date_debut=NULL

    -- si la carte est déja utilisée, alors je lève une exception
    IF ( nb_forfaits > 0) THEN
        RAISE EXCEPTION 'La carte % n\'est pas disponible !!!', NEW.id_carte;
        -- ceci permet de produire un texte d'erreur qui sera affiché si on
        -- essaye d'ajouter un forfait affecté à une carte déja utilisée
    END IF;
    -- si tout se passe bien, alors on retourne NEW qui est la variable contenant
    -- la ligne que l'on est en train de rajouter
   RETURN NEW;
END

$carte_util$ LANGUAGE plpgsql;
```

8.2.2 La déclaration du trigger

Le trigger que l'on veut créer va se déclencher à chaque insertion d'un nouveau forfait. La table sur laquelle va être ce trigger dépend de votre modélisation. Je vous laisse chercher.

La déclaration du trigger va ressembler à :

Si vous voulez remplacer le trigger.

```
DROP TRIGGER IF EXISTS carte_utilisee ON forfait;

CREATE TRIGGER carte_utilisee BEFORE INSERT
    ON la_table_associée_au_trigger FOR EACH ROW
    EXECUTE PROCEDURE carte_utilisee();
```

Cette déclaration signifie : avant l'insertion de lignes dans la table

'la_table_associée_au_trigger' la procédure carte_utilisee() sera exécutée pour chaque nouvelle ligne ajoutée.

Une erreur typique qui se produit lors de l'execution du trigger (pas lors de sa déclaration) :

```
Erreur SQL :
ERROR:  subquery must return only one column
CONTEXT:  SQL statement "SELECT  (SELECT * FROM ...)"
PL/pgSQL function "carte_utilisee" line 21 at assignment
```

Cela se produit lorsque dans votre programme vous voulez affecter dans une variable le résultat d'une requete SQL qui retourne plusieurs champs (au lieu d'un seul). Il vous faut corriger votre fonction puis la recharger.