Gestion des données
TP4 : Bases relationnelles avec PostgreSQL

Olivier Schwander <olivier.schwander@sorbonne-universite.fr>

2021-2022

Ce TP est prévu pour être réalisé dans la machine virtuelle fournie.

Exercice 1 *Accès et chargement des données*

On commence par tester l'accès à la base de données. La commande suivante permet de rentrer interactivement des commandes sql.

```
psql
```

La sortie devrait ressembler à :

```
psql (9.6)
Type "help" for help.

vagrant=#
```

Important Pour le moment, on ne veut pas rentrer interactivement de commandes, donc si la sortie est correcte, taper `ctrl+d` pour quitter le client.

On s'intéresse dans ce sujet à une base de données d'horaires des transports en commun toulousains Tisseo. Les données sont disponibles dans l'archive située à l'adresse http://www-connex.lip6.fr/~schwander/enseignement/m2stat_gd/tisseo.zip.

Extraire l'archive en ligne de commande avec

```
cd ~/Downloads
unzip tisseo.zip
```

Puis changer le répertoire courant du terminal pour aller là où sont les fichiers sql :

```
cd tisseo
ls
```

Le dernier `ls` devrait afficher une série de fichiers `.sql`. On peut voir le contenu d'un fichier avec la commande `cat lefichier.sql`.

Question 1

Regarder le contenu des fichiers `.sql` contenus dans l'archive. Expliquer leur rôle puis exécuter les commandes suivantes pour charger les données dans la base.

```
psql < creabaseTisseo.sql
psql < insertCalendars.sql
psql < insertRoutes.sql
psql < insertStops.sql
psql < insertTrips.sql
psql < insertStopTimes1.sql
psql < insertStopTimes2.sql
```

```
psql < insertStopTimes3.sql
psql < insertStopTimes4.sql
psql < insertStopTimes5.sql
psql < insertStopTimes6.sql
psql < insertStopTimes7.sql
psql < insertStopTimes8.sql
psql < insertStopTimes9.sql
```

Remarque : ces opérations sont à effectuer dans le shell et PAS dans le client SQL (on ne doit donc pas avoir de `vagrant=#` au début des lignes). Cela va prendre un peu de temps et de nombreuses lignes vont s'afficher, c'est normal.

Exercice 2 *Schéma*

Le schéma de la base est le suivant :

- CALENDAR (service_id, monday, tuesday, wednesday, thursday, friday, saturday, sunday, start_date, end_date)
- CALENDAR_DATES (#service_id, date_service, exception_type)
- ROUTES (route_id, route_short_name, route_long_name, route_desc, route_type)
- STOPS (stop_id, stop_code, stop_name, location_type, #parent_station)
- TRIPS (trip_id, #service_id, #route_id, trip_headsign, direction_id)
- STOP_TIMES (#trip_id, #stop_id, stop_sequence, arrival_time, departure_time)

Table CALENDAR calendrier d'activité des services de transport :

— `service_id` : identifiant de service,
— `Monday` : booléen indiquant si le service est actif le lundi,
— `Tuesday` : booléen indiquant si le service est actif le mardi,
— `Wednesday` : booléen indiquant si le service est actif le mercredi,
— `Thursday` : booléen indiquant si le service est actif le jeudi,
— `Friday` : booléen indiquant si le service est actif le vendredi,
— `Saturday` : booléen indiquant si le service est actif le samedi,
— `Sunday` : booléen indiquant si le service est actif le dimanche,
— `start_date` : date de début de mise en service,
— `end_date` : date de fin de mise en service.

Table CALENDAR_DATES calendrier des exceptions de services :

— `service_id` : identifiant du service,
— `date_service` : date du service concerné,
— `exception_type` : identifiant de l'exception.

Table ROUTES les lignes de bus, métro et les navettes :

— `Route_id` : identifiant unique d'une ligne,
— `Route_short_name` : numéro ou nom de ligne de bus ou nom de la navette,
— `Route_long_name` : nom des deux bouts de ligne,
— `Route_desc` : description plus longue de la route,
— `Route_type` : type de la ligne. (0 - Tram, 1 – Metro, 3 – Bus).

Table STOPS tous les arrêts du réseau des bus :

— `Stop_id` : l'identifiant unique d'un arrêt,
— `Stop_code` : un code plus court pour une recherche plus facile pour les utilisateurs,
— `Stop_name` : le nom de l'arrêt,

— `Location_type` : 0 pour un arrêt simple et 1 pour une station plus grande qui contient plusieurs arrêts simples,
— `Parent_station` : identifiant de l'arrêt parent ; tous les arrêts qui ont un arrêt parent ont une valeur d'attribut `Location_type` égale à 0.

Table TRIPS tronçons de deux ou plusieurs arrêts :

— `Trip_id` : identifiant d'un tronçon,
— `service_id` : identifiant d'un service,
— `route_id` : identifiant d'une ligne,
— `trip_headsign` : message d'affichage du tronçon,
— `direction_id` : direction du tronçon (0 aller, 1 retour).

Table STOP_TIMES Indications horaires des tronçons et arrêts :

— `trip_id` : identifiant du tronçon,
— `stop_id` : identifiant d'un aret,
— `stop_sequence` : position de l'arrêt dans le tronçon,
— `arrival_time` : heure d'arrivée à l'arrêt,
— `departure_time` : heure de depart de l'arrêt.

Remarque : on peut utiliser les commandes `\dt` pour visualiser la liste des tables et `\d table` pour visualiser la structure d'une table.

Question 1
Quelles sont les clés primaires ? Expliquer pourquoi les valeurs sont bien uniques.

Question 2
Quelles sont les clés étrangères ? Détailler les liens entre tables.

Exercice 3 *Interrogation*

Question 1
Donner la liste complète des lignes.

Correction

```
SELECT * FROM routes;
```

Question 2
Donner les lignes de métro.

Correction

```
SELECT * FROM routes WHERE route_type=1;
```

Question 3
Donner le numéro et l'intitulé de la ligne correspondant à la route `11821949021891625`.

Correction

```
SELECT route_id, route_short_name, route_long_name, route_type
FROM route
WHERE route_id=11821949021891625;
```

Question 4
Donner, par ordre décroissant de jours hebdomadaires travaillés, les services qui fonctionnent (ou ont fonctionné) le dimanche.

Correction

```
SELECT service_id,
(c.monday+c.tuesday+c.wednesday+c.thursday+c.friday+c.saturday+c.sunday) as nbjour
FROM calendar c
WHERE sunday=1
ORDER BY nbjour;
```

Question 5
Donner le nombre de lignes de tram, de métro et de bus.

Correction

```
SELECT route_type, count(*)
FROM routes
GROUP BY route_type;
```

Question 6
Donner les `trip_headsign` associés aux différentes lignes.

Correction

```
SELECT DISTINCT r.route_long_name, t.trip_headsign, t.direction_id
FROM trips t, routes r
WHERE t.route_id = r.route_id
ORDER BY route_long_name;
```

Exercice 4 *Mise à jour*

Question 1
Insérer dans la table `Calendar_dates` une ligne correspondand à la déclaration d'une exception de type 3 en date du 1er janvier 2014 pour le service 12345679. Que se passe-t-il ?

Correction

```
INSERT INTO Calendar_dates VALUES (12345679,'01-01-2014',3);

--ERREUR à la ligne 1 :
--ORA-02291: violation de contrainte d'intégrité
--(20500153.FK_CALENDARDATES_CALENDAR) - clé parent introuvable
```

Question 2
Le service 4503599629230145 qui circule entre le 2 janvier 2013 et le 31 octobre 2013 circule le dimanche. Mettre à jour la table `Calendar`.

Correction

```
UPDATE calendar
SET sunday=1
WHERE service_id=4503599629230145
AND start_date>='02-01-2013'
AND end_date<='31-10-2014';
```

Question 3
La ligne T2 *Aéroport / Palais de justice* du tramway de Toulouse a été construite depuis l'exportation de la base de données. Ajouter cette ligne dans la table `routes`.

Correction

```
INSERT INTO routes VALUES
(99999999999, 'T2', 'Aéroport / Palais de justice',
'Ligne Aéroport / Palais de justice', 0);
```

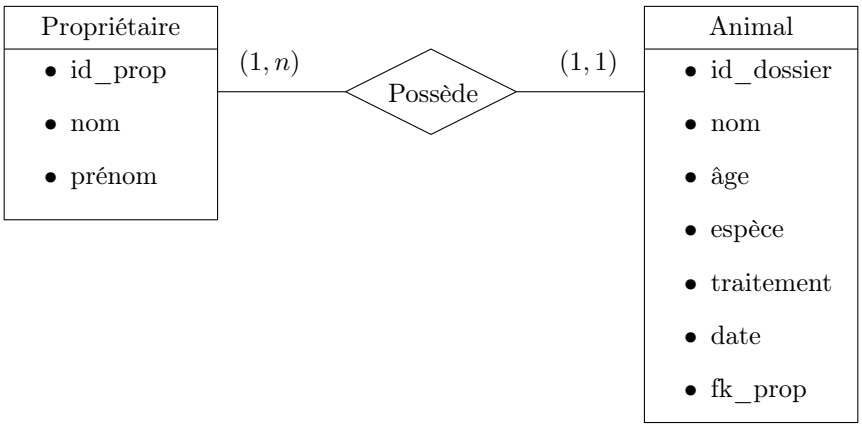
Exercice 5 *Création d'un schéma*

On s'intéresse maintenant à la définition d'un schéma, à partir de la description des données à stocker.

Description de la base Un vétérinaire cherche à stocker des données sur des animaux et leur propriétaire. Les propriétaires sont décrits par leur nom et leur prénom, et peuvent posséder plusieurs animaux. Un animal est décrit par son nom, son âge, son espèce, un numéro de dossier unique de 20 caractères, un booléen indiquant s'il suit un traitement ou non et la date de la dernière visite. Un animal n'est possédé que par un seul propriétaire.

Question 1
Dessiner le schéma entité-association correspondant à cette description.

Correction



Question 2
Donner un exemple d'entrées dans les différentes tables de cette base.

Correction

Propriétaire
- Identifiant : 1
- Prénom : Tintin
- Nom : Hergé

Animal
- Dossier : ABCDEFGHIJKLMNOPQRST
- Nom : Milou
- Âge : 77 ans
- Espèce : probablement un fox-terrier à poil dur dont la blancheur de la robe est inhabituelle
- Traitement : non

— Date : 1929-01-10
— Propriétaire : 1

Question 3

Propriétaire L'identifiant sera un entier, les noms et prénoms seront du type `VARCHAR(beaucoup)` ou de type `TEXT` pour des noms de longueur arbitrairement grande (plus compliqué à gérer pour le serveur).

Animal Le numéro de dossier est par définition une chaîne de

20 caractères de long `CHAR(20)`. Le nom de l'animal sera `VARCHAR(beaucoup)` ou `TEXT`. L'âge un `INTEGER` correspondant au nombre d'années (on pourrait compter plus précisément en mois, mais un entier est ici le plus naturel). L'espèce est a priori quelque chose d'assez court donc `VARCHAR(pas trop)` est satisfaisant (le plus logique sera d'avoir une troisième table avec les espèces, et donc une clé étrangère ici). La date est au format `DATE` ou `DATETIME` si on veut conserver l'heure du rendez-vous (si on voulait un historique plus détaillé, il faudrait plutôt faire une table des rendez-vous). La colonne traitement est naturellement de type `BOOLEAN`.

Correction

Voir question suivante.

Pour réaliser l'implantation de ce schéma, il faut commencer par créer une nouvelle base de données, nommée cette fois `veto` :

```
createdb veto
```

On accédera à cette base avec le client de la façon suivante :

```
psql veto
```

Question 4

Créer les tables nécessaires à l'aide d'une séquence de requêtes `CREATE`.

Correction

L'ordre est important à cause de la contrainte de clé étrangère.

```
CREATE TABLE proprietaire (
  id_prop INTEGER PRIMARY KEY,
  nom VARCHAR(128),
  prenom VARCHAR(128),
);

CREATE TABLE animal (
   id_dossier CHAR(20) PRIMARY KEY,
   nom VARCHAR(128),
   age INTEGER,
   espece VARCHAR(64),
   traitement BOOLEAN,
   date DATE,
   id_prop INTEGER
   CONSTRAINT fk_propriétaire FOREIGN KEY (id_prop) REFERENCES proprietaire(id_prop),
);
```

Question 5
Insérer dans la base les données de votre exemple.

Correction

L'ordre est important à cause de la contrainte de clé étrangère. Il n'y a pas besoin de préciser l'ordre des colonnes, puisqu'on les remplit toutes en même temps.

```
INSERT INTO proprietaire VALUES (1, 'Tintin', 'Hergé');
INSERT INTO animal
VALUES (ABCDEFGHIJKLMNOPQRST, 'Milou', 77, 'chien', false, '1929-01-10', 1);
```

Question 6
Écrire une requête pour récupérer la liste des animaux suivant un traitement, avec le nom de leur propriétaire.

Correction

```
SELECT a.nom, p.nom, p.prenom
FROM animal a, proprietaire p
WHERE a.id_prop = p.id_prop AND a.traitement = true;
```