

TP de BD avancées n° 4

Preliminaires

Organisation de la base de données PostgreSQL Vous avez accès à une base commune à tous les utilisateurs qui s'appelle **base**. À chaque login correspond un nom d'utilisateur : si votre login est **toto**, vous serez l'utilisateur **toto** sur la base. Chaque utilisateur a un schéma de même nom que son login. Par défaut, vous ne pouvez créer des tables que sur ce schéma. Vous aurez, par ailleurs, accès en lecture sur le schéma **ratp** que nous allons utiliser dans ce TP.

Le schema **public** qui est normalement le schéma par défaut, vous est également accessible en lecture seule.

Utiliser la base de données La base de données PostgreSQL est sur **nivose**, vous devez donc vous logger sur cette machine. Pour accéder à cette base, tapez : **/usr/local/pgsql/bin/psql base nom_login**. (Modifiez vos fichier de configuration, pour ne plus avoir à retaper le chemin complet de la commande à chaque fois.)

Pour configurer le choix du langage procédural de votre base, vous devez entrer (la première fois uniquement) :

```
CREATE PROCEDURAL LANGUAGE plpgsql;
```

Commandes pratiques en psql

- **\?** donne la liste de toutes les commandes psql.
- **\h <commande SQL>** donne une description de la commande SQL.
- **\d**
 - employé seul, donne la liste des tables,
 - suivi du nom d'un objet donne une description de cet objet, par exemple, **\d ratp.trips** donne la liste des attributs, contraintes, etc. de la table **trips** du schéma **ratp**.
 - suivi du nom d'un schéma avec un point accolé au nom du schéma, donne la liste détaillée des tables et index du schéma
- **\i nomfichier** exécute les commandes dans le fichier **nomfichier.sql**.

N'oubliez pas de taper **;** après chaque ordre SQL...

Pour utiliser une table d'un schéma, préfixez son nom par le nom du schéma, par exemple **ratp.calendar** pour la table **calendar** du schéma **ratp**.

Pour voir les tables des autres schémas que le vôtre et **public** : **SET search_path TO sch1, sch2, ... ;**. (À faire à chaque nouvelle session.)

Les tables du schéma **sch1** seront alors utilisées en priorité lorsque vous ferez une sélection, écriture... Elles seront aussi visibles en priorité lorsque vous ferez **\d** sous **psql** pour avoir la liste des tables.

Suppression d'une fonction Sous `psql`, pour obtenir la liste de toutes les surcharges de la fonction `nomfonction` (avec la liste du type de ces paramètres), tapez `\df nomfonction`. Pour supprimer une fonction, ou une de ses surcharges, tapez

```
DROP FUNCTION nomfonction(type1, type2, ...);
```

Les tables du schéma `ratp` La signification des tables n'est pas tout à fait celle que l'on a vu en TD. Les clefs primaires sont soulignées, les clefs étrangères sont indiquées par un #.

- `calendar(service_id, start_date, end_date)`
- `calendar_dates(service_id#, date)`
- `routes(route_id, route_short_name, route_long_name, route_type, route_color, route_text_color)`
- `stops(stop_id, stop_name, stop_desc, stop_lat, stop_lon)`
- `transfers_sans_doublons(from_stop_id#, to_stop_id#, min_transfer_time)`
- `trips(route_id#, service_id#, trip_id, trip_short_name, direction_id)`
- `stop_times(trip_id#, arrival_time, departure_time, stop_id, stop_sequence, stop_headsign)`

Explication complémentaire Un service est un ensemble de dates; on donne dans `calendar` la plage de dates qui les englobent (par exemple, du 2 septembre au 18 novembre), et dans `calendar_dates` les dates effectivement concernées, par exemple, tous les dimanches de la plage précédente plus le 1er et le 11 novembre.

Une journée donnée sera concernée par plusieurs services.

La table `transfers` contenant des lignes en doubles, la table `transfers_sans_doublons` a été créée pour obtenir un schéma plus cohérent.

Ces tables sont très volumineuses, NE LES COPIEZ PAS SUR VOTRE SCHÉMA.

Nous allons dans ce TD utiliser deux tables : `ratp.calendar` et `ratp.calendar_dates`.

Méthodologie de travail Pour chaque TP, créez et éditez un script qui exécute les exercices demandés, plutôt que de taper les commandes directement sous `psql`. Vous gagnerez du temps, mais vous aurez aussi une solution de votre TD qui pourra vous être demandée en fin de séance.

Conversion de type et fonctions sur les dates Plusieurs fonctions prédéfinies en Postgres permettent de manipuler les données de type `DATE`. On peut par exemple ajouter un entier n à une date pour obtenir la date n jours plus tard, ou soustraire une date à une autre pour connaître le nombre de jours entre les deux dates.

Un entier peut être converti en une chaîne à l'aide de la fonction `to_char(nombre_a_convertir, format)` où `format` est une chaîne de la forme `'99999999'` qui indique la longueur de la chaîne que l'on souhaite. Une chaîne peut être convertie en type `DATE` avec la fonction `to_date(chaine_a_convertir, format)`, où `format` est une chaîne de la forme `'YYYYMMDD'` qui indique le format dans lequel est donnée la date.

Travail à faire

Exercice 1 Écrivez une fonction `duree_intervalle` qui prend en entrée un `service_id` et qui retourne le nombre de jours entre `ratp.calendar.start_date` et `ratp.calendar.end_date`.

Testez-la, par exemple, en tapant

```
select service_id, duree(service_id) from ratp.calendar where service_id <1164400;
```

Exercice 2 Écrivez une fonction qui prend en entrée un `service_id` et retourne le nombre de jours correspondant à ce service dans la table `ratp.calendar_dates`.

Exercice 3 Où est ma fonction ? Les tables système contiennent les meta-données de la base de données. Consultez la liste des attributs de la table `pg_proc` pour voir comment votre fonction est stockée sous Postgres. Exécutez la requête suivante.

```
SELECT proname, prosrc
FROM pg_proc
WHERE proname = 'duree_intervalle';
```

Exercice 4 Écrivez une fonction qui parcourt la table `ratp.calendar` et incrémente un compteur si pour un `service_id` donné, le nombre de jours dans la table `calendar_dates` dépasse le nombre de jours dans l'intervalle dans la table `calendar`. La fonction retourne la valeur du compteur à la fin.

Exercice 5 Nous allons voir comment se compare l'efficacité de la fonction précédente en comparaison à une requête SQL qui fait le même travail. La commande `\timing` permet d'afficher le temps, en millisecondes, pris par une requête. Tapez `\timing` sous `psql`. Exécutez votre fonction, notez le temps d'exécution. Puis écrivez une requête SQL qui fait le même travail, et mesurez le temps pris par la requête.