Bases de données Avancées
TP n° 5 : Fonctions en pl/pgslq

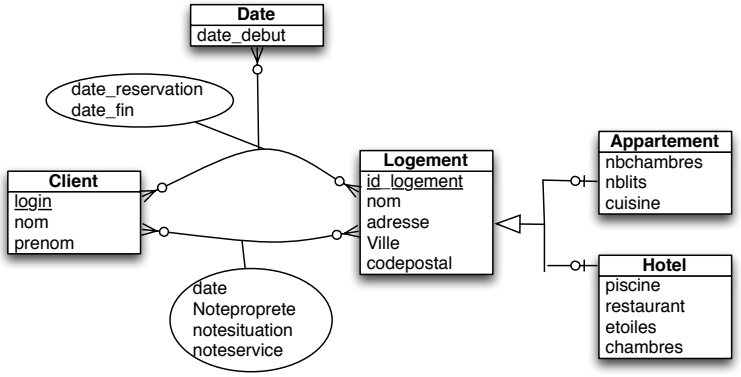
Rappel sur l'utilisation de psql sur nivose

Comme nous l'avons vu au TP 2, pour l'essentiel vous devez faire :

- `ssh login@nivose.informatique.univ-paris-diderot.fr`
- `psql -d base_m1 login`
- `set search_path to tp_login;`

Les hôtels

Les requêtes portent sur une base de données d'hôtels que les utilisateurs peuvent réserver et sur lesquels les utilisateurs peuvent donner des notes. Les hôtels sont soit des hôtels traditionnels ou des studios ou appartements meublés en location pour de courts séjours. Les clients peuvent réserver un logement et peuvent noter les logements.



Les tables sont structurées comme suit :

```
client = (login, nom, prenom)
logement = (numlogement, nom, adresse, ville, codepostal)
hotel = (numlogement, piscine, restaurant, etoiles, chambres)
appartement = (numlogement, nbchambres, nblits, cuisine)
reserve = (login, numlogement, datedebut, datefin, datereservation)
avis = (login, numlogement, date, noteproprete, noteservice, notesituation)
```

Vous pouvez installer les données correspondante en important le fichier fourni dans moodle. A tout moment vous pouvez obtenir les détails des tables en tapant `\d nomtable`.

Exercice 1 :

Donnez les requêtes SQL permettant de trouver les informations suivantes. Portez une attention particulière aux données nulles ou inexistantes.

1. Donner tous les logements qui n'apparaissent ni en tant qu'hôtel, ni en tant qu'appartement.

2. Écrire une requête qui retourne, pour chaque logement, son nom, et un attribut `est_hotel` qui est `TRUE` si le logement donné en paramètre est un hôtel, `FALSE` si c'est un appartement, et `NULL` si ce n'est ni l'un ni l'autre.
3. Donner la note moyenne de chaque logement sur chaque critère. Pour chaque hôtel, affichez les trois moyennes (une moyenne par critère) ainsi qu'une moyenne générale, avec un coefficient de 1/3 sur chacun des trois critères. (Que s'affiche-t-il si une des moyennes est NULL ?)
4. Afficher le nom des hôtels dont le nombre de réservations pour aujourd'hui dépasse le nombre de chambres.

PL/pgSQL

Rappelons la syntaxe d'une fonction PL/pgSQL.

```
CREATE OR REPLACE FUNCTION mafonction(...) RETURNS ... AS $$
DECLARE
    ...
BEGIN
    ...
END;
$$ LANGUAGE plpgsql;
```

Pour l'appeler il suffit de faire `SELECT mafonction(...);`

> Indication : lorsque vous utilisez PL/pgSQL, il faut résister à toute tentation de remplacer par du code PL/pgSQL ce qui peut être réalisé par une bonne requête SQL. N'oubliez pas que le moteur d'optimisation SQL est très efficace et qu'une bonne requête SQL est toujours plus performante que du code PL/pgSQL.

Exercice 2 :

Écrire une fonction booléenne `est_hotel` qui retourne `TRUE` si le numéro de logement donné en paramètre est un hotel, `FALSE` si c'est un appartement, et `NULL` si c'est ni l'un ni l'autre. Si le logement n'existe pas, terminer avec une exception `RAISE 'Logement inexistant % ', num_logement USING ERRCODE='20002'`. (Voir Exercice 1 Question 2.) Vérifier les résultats avec `SELECT nom, est_hotel(numlogement) FROM logement;` Puis pour vérifier qu'un logement inexistant provoque bien une erreur `SELECT est_hotel(XXX);`, en remplaçant `XXX` par un numéro qui ne correspond pas à un logement.

Exercice 3 :

Écrire une fonction `reserver` qui prend en paramètre un numéro de logement `num_logement`, une date d'arrivée `date_debut` et de départ `date_fin` et crée une réservation pour `CURRENT_USER`. Si ce client n'existe pas, terminer avec une exception `RAISE 'Client inexistant % ', CURRENT_USER USING ERRCODE='20001'`. (Puis ajoutez-vous à la table `client`.) Si le logement n'existe pas, terminer avec une exception `RAISE 'Logement inexistant % ', num_logement USING ERRCODE='20002'`.

Exercice 4 :

Écrire une fonction booléene `appartement_disponible` qui vérifie si un appartement est disponible aux dates demandées. Si le logement n'est pas un appartement, terminer avec une exception `RAISE 'Logement % n''est pas un appartement', num_logement USING ERRCODE='20101';` Indication : il suffit de vérifier qu'aucune réservation ne chevauche l'intervalle de dates demandé.

Exercice 5 :

Écrire une fonction booléene `hotel_disponible` qui vérifie si un hôtel est disponible à une date donnée. Si le logement n'est pas un hotel, terminer avec une exception `RAISE 'Logement % n''est pas un hotel', num_logement USING ERRCODE='20102';`

Exercice 6 :

Écrire une fonction `disponible` qui vérifie si un logement est disponible aux dates demandées. Pour les appartements, appeler la fonction `appartement_disponible` sur l'intervalle, et pour les hotels, appeler la fonction `hotel_disponible` sur chaque jour de l'intervalle.

Exercice 7 :

Reprendre la fonction `reserver` et intégrer la validation des dates. Si les dates sont invalides, terminer avec une exception `RAISE 'Reservation impossible pour les dates % % ', date_debut, date_fin USING ERRCODE='20003'`. Si tout se passe correctement, terminer avec `RAISE NOTICE 'Reservation faite.'`

Exercice 8 :

Écrire une fonction booléenne `reservation_sure` qui prend en paramètre le nom et prénom de client, date d'arrivée, date de départ, le numéro de logement.
Si la table `client` ne contient pas de client avec le même nom et prénom alors la fonction ajoutera le nouveau client dans la table `client`, comme son `login` on prendra le mot obtenu en concaténant son nom et prénom. (Rappel : `a||b` donne la concaténation de `a` et `b`.) Si la table `client` contient plusieurs clients ayant le même nom et prénom alors on lance une exception.
Si la table client contient exactement un client avec le même nom et prénom alors on suppose que c'est le client qui fait la réservation.
La réservation est impossible s'il existe un jour dans la période donnée tel que

- il n'y a aucune chambre libre si le logement est un hôtel ou
- il existe un jour tel que l'appartement est réservé si le logement est un appartement.

Si la réservation est impossible lancer une exception.
Sinon faites la réservation.