Utilisation du shell Unix
10 - Shell et bases de données

1 Utilisation du client PostgreSQL

On a vu qu'on pouvait réaliser certaines fonctions des bases de données (sélection, projection) grâce aux outils Unix de manipulation du texte (`grep`, `cut`), mais un SGBD reste évidemment préférable pour des données ou des opérations plus complexes.

Pour les TP de bases de données, vous avez utilisé le système de gestion de bases de données (SGBD) PostgreSQL. Ce SGBD est installé sur le serveur `gigondas` et est accessible par le web avec l'outil `dbeaver` :

```
dbeaver &
```

Le SGBD PostgreSQL est aussi accessible avec le *client PostgreSQL* : il s'agit d'une commande Unix qui se comporte comme un shell : elle affiche un *prompt*, lit des commandes, les exécute et affiche les résultats. Cette commande est installée sur tous les postes Linux des salles de TP.

1. Lancez la commande `psql` avec les options permettant l'accès à votre base sur le serveur `gigondas` :
   `psql -h gigondas -U` *`login login`*
   puis donner le mot de passe (`login`, à moins que vous ne l'ayez changé).
2. Vous devez alors pouvoir lancer une requête SQL ou une commande PostgreSQL. Les commandes PostgreSQL commencent par \. Essayez par exemple `\?`.
   Ça affiche l'aide de toutes les commandes avec la commande `less` de Linux. Vous pouvez donc quitter l'aide avec `q` ou avancer d'une ligne avec ↩ ou d'une page avec SPACE.
3. Essayez `\l` qui affiche la liste des bases de données.
4. Vous avez accès (en lecture) à la base de données `bdvins`, qui est une base exemple pour les TP ; pour l'utiliser, il faut changer de base :
   `\connect bdvins` ou simplement `\c bdvins`
   Vous pouvez avoir la liste des tables de cette base avec : `\dt`
   Vous pouvez aussi simplement lancer des requêtes `SELECT` sur les tables, essayez par exemple : `select * from couleurs;`
5. Vous pouvez revenir à votre base de données personnelle avec : `\c` *`login`*
6. Enfin, vous pouvez quitter le client PostgreSQL avec : `\q`

Récupération des fichiers fournis :
On vous donne dans `/users/but/info/Public` une archive `TP-Shell-SQL.tgz` contenant :

— `village.sql`, fichier de commandes SQL pour la création des tables du *Village gaulois* ;
— `PGPass`, modèle du fichier de configuration du client PostgreSQL (voir section suivante) ;
— `Requetes`, un dossier contenant quelques requêtes très simples.

Récupérez cette archive et extrayez-là (par exemple avec votre script `recup`). L'extraction crée un répertoire `TP-Shell-SQL` ; placez-vous dans ce répertoire pour la suite du TP.

2 Simplification de l'accès

La commande de lancement du client PostgreSQL de la section précédente est un peu longue à saisir mais on peut la simplifier puisque si vous ne précisez pas l'utilisateur alors il utilisera l'utilisateur courant comme valeur par défaut. Si vous êtes connecté en `martin` alors :

`psql -h gigondas` ⟺ `psql -h gigondas -U martin martin`

Mais il faut quand même donner le mot de passe à chaque fois. On peut enregistrer le mot de passe pour une paire (*host*, *user*) dans un fichier baptisé `.pgpass` qui doit être rangé dans votre répertoire de travail par défaut (*HOME directory*).

7. Avec votre éditeur de textes favori, modifiez le fichier `PGPass` fourni pour qu'il utilise votre base de données (il suffit de remplacer `user` et `password` par *votreLogin*).
8. Copiez ce fichier dans votre *HOME directory* en le renommant `.pgpass`
9. Assurez-vous que vous seul avez accès au fichier en lecture. S'il le faut changez les droits avec la commande `chmod`.
10. Vous devriez maintenant pouvoir vous connecter à votre base en tapant simplement :
    `psql -h gigondas`
11. Comme on n'est jamais assez fainéant, créez un alias, par exemple `gsql` (`g` pour `gigondas`) :
    `alias gsql='psql -h gigondas'`
12. Pour que cet alias soit permanent, ajoutez-le à votre fichier `$HOME/.bash_aliases` qui est automatiquement exécuté par le fichier `$HOME/.bashrc` à chaque lancement d'un shell.

Dans la suite, je suppose que vous pouvez vous connecter à votre base PostgreSQL en tapant simplement : `gsql`

3 Exécution de requêtes depuis des fichiers

Le principal intérêt du client PostgreSQL est qu'il permet très facilement d'exécuter une requête contenue dans un fichier, voire d'enchaîner l'exécution de plusieurs requêtes grâce à un script shell. Nous allons voir comment dans cette section.

Tables du village gaulois :
Si vous ne l'avez pas déjà fait en TP de *Bases de données*, vous devez créer les tables du village gaulois dans votre base personnelle, il suffit pour cela de demander au serveur PostgreSQL d'exécuter les requêtes de création contenues dans le fichier fourni `village.sql`.

13. Placez-vous dans le répertoire `TP-Shell-SQL` puis lancez `gsql` ;
14. La commande `\i` du client PostgreSQL permet d'exécuter les requêtes contenues dans un fichier. Vous pouvez donc créer les tables en lançant : `\i village.sql`
    NB : bien entendu, si vous aviez déjà créé ces tables en TP *Bases de données*, vous aurez des messages d'erreur.
15. Si tout s'est bien passé, vous devriez voir les tables installées en utilisant la commande PostgreSQL : `\dt`
16. Le répertoire `Requetes` contient 3 requêtes simples, dans 3 fichiers différents : `listeBatailles.sql`, `listeHabitants.sql` et `quiMangeQuoi.sql`
    Avec la commande `\i`, exécutez successivement chacune de ces requêtes.
17. Quittez le client PostgreSQL.

La commande `psql` lit ses commandes au clavier, c'est-à-dire sur son *entrée standard*, or on a vu qu'on pouvait rediriger l'entrée standard d'une commande sur le contenu d'un fichier, avec `'<'`.

18. Testez la commande suivante sur le fichier `listeHabitants.sql` du répertoire `Requetes` :
    `gsql < Requetes/listeHabitants.sql`
19. Testez la commande `gsql` sur les deux autres fichiers du répertoire `Requetes`.

Pratique non ? Notez qu'on aurait aussi pu créer les tables avec `gsql < village.sql` ; on peut même les effacer avec le script SQL fourni `effaceVillage.sql`.

4 Retour au shell Unix

L'intérêt principal de l'utilisation du client PostgreSQL par rapport à DBeaver est qu'on peut « scripter » les commandes, c'est-à-dire créer des scripts qui enchaînent l'exécution de plusieurs requêtes et récupérer leurs résultats.

On souhaite créer un compte-rendu de l'exécution de plusieurs requêtes. On suppose que chaque requête est contenue dans un fichier avec l'extension `.sql`, par exemple `listeHabitants.sql`. Pour chaque requête on veut afficher le nom du fichier, son contenu (texte de la requête) et le résultat de l'exécution, donc quelque chose comme sur la figure 1.

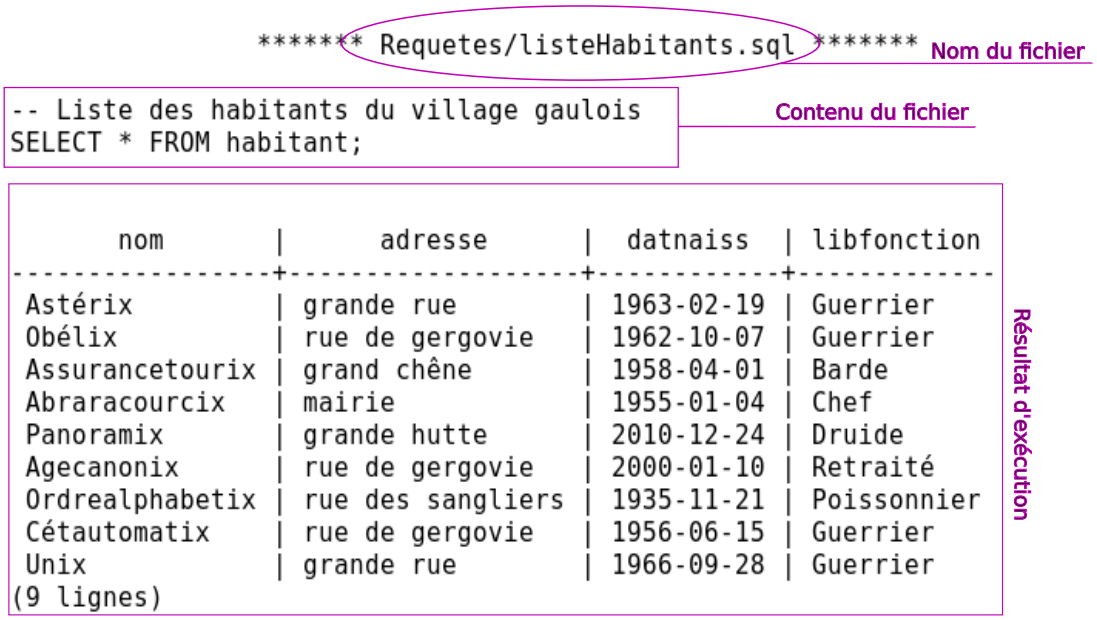

FIGURE 1 – Exemple de compte-rendu pour une requête

4.1 Automatiser les comptes-rendus

NB : placez les scripts dans votre répertoire `$HOME/bin` pour qu'ils soient accessibles de partout.

20. Écrivez un script `crSQL` qui affiche quelque chose de semblable à la figure 1 pour un script dont le nom sera passé en paramètre. Le texte de la figure 1 sera par exemple obtenu avec :
    ```
    crSQL listeHabitants.sql
    ```
    NB :
    - mettez vos noms en commentaire au début du script (pour le compte-rendu) ;
    - les *alias* ne fonctionnent pas dans les scripts. Vous devrez donc utiliser la commande `psql -h gigondas`.
21. Écrivez un script `crToutSQL` qui affiche la même chose pour chaque fichier ayant l'extension `.sql` **dans le répertoire courant**. Testez ce script dans votre répertoire `Requetes`.
22. Pour obtenir un fichier texte (imprimable), il suffit de rediriger la sortie du script dans un fichier, par exemple `CR-SQL.txt`. Testez cette fonctionnalité dans votre répertoire `Requetes`.

4.2 Description des tables

On voudrait maintenant obtenir la description de toutes les tables de notre base de données (une fois imprimée, cette description facilite le travail de création de requêtes). Il faut procéder en deux étapes :

i) obtenir la liste des tables (`\dt`) ;

ii) pour chaque table afficher la description (`\d table`).

23. Pour éviter de créer un fichier SQL pour chaque requête, on peut passer la requête en paramètre au client PostgreSQL grâce à l'option `-c`. Exemple : `gsql -c "\\dt"`
Il faut doubler le `'\'` car il est interprété par le shell ou alors le masquer en utlilisant des *quotes* : `gsql -c '\dt'`

24. Comme on ne veut que la liste des tables, il faut éviter d'afficher la première ligne (`Liste des relations`) ainsi que l'entête des colonnes et le total à la fin. Il faut passer le client en mode « tuples seulement » avec la commande `\t` . Essayez par exemple :
```
gsql -c '\t' -c '\dt'
```

25. Pour obtenir une liste des tables « propre », c'est-à-dire facilement réutilisable dans un script shell, on peut annuler l'alignement des colonnes avec l'option `-A` :
```
gsql -A -c '\t' -c '\dt'   ⟹   L'affichage des tuples seuls est activé.
                               public|bataille|table|volfoni
                               ...
                               public|potion|table|volfoni
                               public|prendre|table|volfoni
```

26. Mais PostgreSQL affiche en tête une liste pour nous rappeler qu'on a activé `\t` : `"L'affichage des tuples seuls est activé."` Pour éliminer cette ligne on peut utiliser la commande `tail` qui affiche les dernières lignes d'un fichier (ou de son entrée standard) :
```
gsql -A -c '\t' -c '\dt' | tail -5
```
affichera les 5 dernières lignes du résultat de `gsql` .
On peut aussi afficher toutes les lignes à partir de la 3e :
```
gsql -A -c '\t' -c '\dt' | tail +3
```

27. Écrivez le script `mesTables` réalisant l'affichage de la description de toutes vos tables. On aura soin d'afficher le nom de la table avant sa description, par exemple comme ça :
```
*** potion ***
```
Pour la table `potion`, vous afficherez donc :
```
*** potion ***
                          Table « public.potion »
  Colonne  |         Type          | Collationnement | NULL-able | Par défaut
-----------+-----------------------+-----------------+-----------+------------
 nompotion | character varying(20) |                 | not null  |
Index :
    "potion_pkey" PRIMARY KEY, btree (nompotion)
Référencé par :
    TABLE "prendre" CONSTRAINT "prendre_nompotion_fkey" FOREIGN KEY (nompotion)
        REFERENCES potion(nompotion)
```

Compte-rendu

Vous rendrez un listing des scripts `crSQL`, `crToutSQL` et `mesTables`. ainsi que du fichier `CR-SQL.txt` de la question 22. Pensez à ajouter vos noms en commentaire dans chaque script et à imprimer en une seule fois pour économiser le papier. Par exemple si vous êtes dans le répertoire `Requetes` :

```
imprime ~/bin/crSQL ~/bin/crToutSQL CR-SQL.txt ~/bin/mesTables
```

N'hésitez pas à copier-coller cette commande pour gagner du temps.