Bases de données relationnelles PostgreSQL sous Linux

Table des matières

Objectifs

- Savoir utiliser le client PostgreSQL *psql* pour créer et gérer des bases de données
- Savoir installer le serveur et le client PostgreSQL sous Linux (en complément)

Introduction

Ce module est destiné a donner les éléments de base permettant d'utiliser un client *psql* déjà installé sur une machine Linux pour se connecter à un serveur *postgresql* déjà installé.

En complément, le module expose les opérations de base permettant d'installer le client et le serveur PostgreSQL sur une machine Linux.

Cours

1. Introduction à PostgreSQL : présentation, installation et utilisation du client

1.1. Présentation de PostgreSQL

PostgreSQL est :

- un SGBDR
- libre (licence BSD)
- multi-plate-formes (Unix, Linux, Windows, MacOS, ...)
- puissant
- très respectueux du standard
- très bien documenté

Documentation de PostgreSQL *Fondamental*

- www.postgresql.org/docs[1]
- en français : docs.postgresqlfr.org[2]

Fonctionnement général

PostgreSQL comme la grande majorité des SGBD est un logiciel *client∗-serveur∗*. Il faut donc pour l'utiliser un serveur (programme *postgresql* sous Linux) et un client.

Il existe plusieurs clients, les deux plus utilisés sont le client textuel *psql* et le client graphique *PgAdmin III*.

Complément

- www.postgresql.org[3]
- www.postgresql.fr[4]

1.2. Le client textuel "psql"

psql *Définition*

`psql` est le client textuel de PostgreSQL.

[1] https://www.postgresql.org/docs/current
[2] https://docs.postgresqlfr.org/current/
[3] http://www.postgresql.org/
[4] http://www.postgresql.fr/

Connexion

Méthode

```
psql -h tuxa.sme.utc -U nf17a001 -d dbnf17a001
```

Cette commande *psql* essaye de se connecter sur la machine *tuxa.sme.utc* avec un utilisateur PostgreSQL nommé *nf17a001* à la base de données *dbnf17a001*.

- Un serveur PostgreSQL doit tourner sur la machine distante *tuxa.sme.utc* (sur le port standard 5432).
- Un utilisateur *nf17a001* doit exister sur PostgreSQL et avoir un mot de passe défini.
- Une base de données *dbnf17a001* doit exister sur PostgreSQL et l'utilisateur *nf17a001* doit avoir le droit d'y accéder.
- Le mot de passe de l'utilisateur *nf17a001* sera demandé.

Écrire une instruction SQL

Syntaxe

```
dbnf17p015=> SELECT * FROM matable ;
```

Écrire une instruction SQL sur plusieurs lignes

Syntaxe

Une instruction SQL peut s'écrire sur une ou plusieurs lignes, le `retour chariot` n'a pas d'incidence sur la requête, c'est le `;` qui marque la fin de l'instruction SQL et provoque son exécution.

```
dbnf17p015=> SELECT *
dbnf17p015-> FROM matable
dbnf17p015-> ;
```

On notera dans `psql` la différence entre les caractères => et -> selon que l'on a ou pas effectué un retour chariot.

Commandes de base : aide

Fondamental

`\?` : Liste des commandes `psql`

`\h` : Liste des instructions SQL

`\h CREATE TABLE` : Description de l'instruction SQL `CREATE TABLE`

Commandes de base : catalogue

Fondamental

`\d` : Liste des relations (catalogue de données)

`\d maTable` : Description de la relation `maTable`

Commandes de base : quitter

Fondamental

`\q` : Quitter `psql`

Complément

http://www.postgresql.org/docs/current/static/app-psql.html

2. Éléments de base pour l'utilisation de PostgreSQL

2.1. Exécuter des instructions SQL depuis un fichier

Il est souvent intéressant d'exécuter un fichier contenant une liste de commandes SQL, plutôt que de les entrer une par une dans le terminal. Cela permet en particulier de recréer une base de données à partir du script de création des tables.

Syntaxe

Pour exécuter un fichier contenant du code SQL utiliser la commande PostgreSQL `\i chemin/fichier.sql`

- `chemin` désigne le répertoire dans lequel est le fichier `fichier.sql`
- le dossier de travail de `psql` est le dossier dans lequel il a été lancé, le script peut être lancé à partir de son dossier `home` pour en être indépendant (`~/.../fichier.sql`)
- chaque commande doit être terminée par un `;`

```
dbnf17p015=> \i /home/me/bdd.sql
```

Programmer une base de données avec PostgreSQL ***Méthode***

Pour programmer une base de données sous PostgreSQL, voici une méthode générale :

- Rendez-vous dans un répertoire de travail : `cd /home/me/bdd1`
- Créez ou ouvrez un fichier texte avec un éditeur colorisant : `atom fichier.sql`
- Ouvrez un terminal et exécutez votre client *psql*
- Écrivez votre code SQL, et testez-le au fur et à mesure : `\i fichier.sql`

Tester votre code régulièrement ***Conseil***

Afin de tester régulièrement votre base de données, pensez à insérer en début de script des instruction de destruction des tables.

On supprime les tables dans l'ordre inverse de leur création, on peut ajouter la clause `IF EXISTS` afin d'éviter les erreurs lorsqu'une exécution précédente avait déjà échoué.

```
DROP TABLE IF EXISTS t2 ;
DROP TABLE IF EXISTS t1 ;
CREATE TABLE t1 (a VARCHAR PRIMARY KEY);
CREATE TABLE t2 (a VARCHAR REFERENCES t1(a));
```

2.2. Types de données

Types standards

On retrouve dans PostgreSQL la plupart des types standard SQL :

- numériques : integer (smallint, bigint), real (double precision)
- dates : date (time, timestamp)
- chaînes : char, varchar
- autres : boolean

Autres types

On notera les autres types suivants (non standard) :

- identifiant : serial, uuid
- numériques : money
- chaînes : text
- non SQL : xml, json

Fonctions de formatage des types de données *Complément*

https://docs.postgresql.fr/current/functions-formatting.html

Documentation *Complément*

https://docs.postgresqlfr.org/current/datatype.html

2.3. Importer un fichier CSV sous PostgreSQL

Syntaxe

```
\copy nom_table (att1, att2, ...) FROM 'fichier.csv' WITH CSV DELIMITER ';' QUOTE '"'
```

- `WITH` introduit les options de l'import
- `CSV` indique qu'il s'agit d'un fichier CSV
- `DELIMITER 'c'` indique que le caractère **c** est utilisé comme délimiteur de champ (en général **;** ou **,**)
- `QUOTE 'c'` indique que le caractère **c** est utilisé comme délimiteur de chaîne (en général **"**)

Remarque

- La table `nom_table` doit déjà exister
- Le nombre de colonnes spécifié doit correspondre au nombre de colonnes du fichier CSV
- Les types doivent être compatibles

Remarque

Ajouter l'option `HEADER` après `WITH CSV` si le fichier CSV contient une ligne s'entête.

```
\copy nom_table (att1, att2, ...) FROM 'fichier.csv' WITH CSV HEADER DELIMITER ';' QUOTE '"'
```

Localisation du fichier CSV depuis psql *Méthode*

Par défaut, la commande `\copy` prendra le chemin du répertoire courant au moment où la commande `psql` a été lancée.

Sous `psql`, vous pouvez utiliser les commandes :

- `dbnf17p007=> \! pwd`

  Pour exécuter la commande *shell* `pwd` et obtenir le répertoire courant
- `dbnf17p007=> \cd directory`

  Pour changer le répertoire courant

Complément

PostgreSQL sous Linux (cf. p.11)

Fichier CSV (cf. p.24)

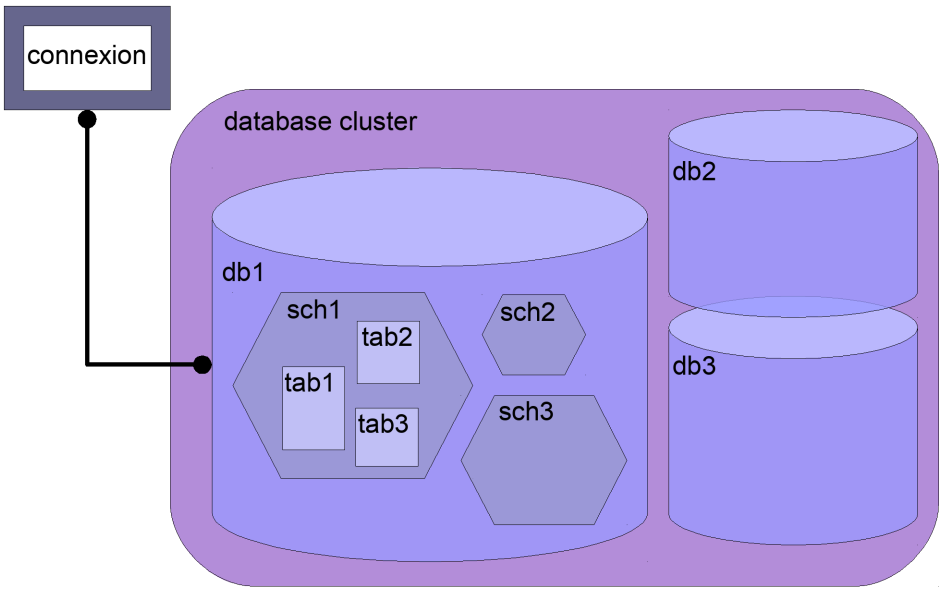
2.4. Notion de schéma sous PostgreSQL

Définition

« A PostgreSQL database cluster contains one or more named databases. Users and groups of users are shared across the entire cluster, but no other data is shared across databases. Any given client connection to the server can access only the data in a single database, the one specified in the connection request.

A database contains one or more named schemas, which in turn contain tables. Schemas also contain other kinds of named objects, including data types, functions, and operators. The same object name can be used in different schemas without conflict; for example, both schema1 and myschema can contain tables named mytable. Unlike databases, schemas are not rigidly separated: a user can access objects in any of the schemas in the database he is connected to, if he has privileges to do so.

http://www.postgresql.org/docs/8.4/static/ddl-schemas.html »



Organisation en cluster, base, schéma, table dans PostgreSQL

Créer un schéma

Syntaxe

```
1 CREATE SCHEMA myschema;
```

Créer une table dans un schéma

§ *Syntaxe*

```
CREATE TABLE myschema.mytable (
...
);
```

Requêter dans un schéma

§ *Syntaxe*

```
SELECT ...
FROM myschema.mytable
```

? *Exemple*

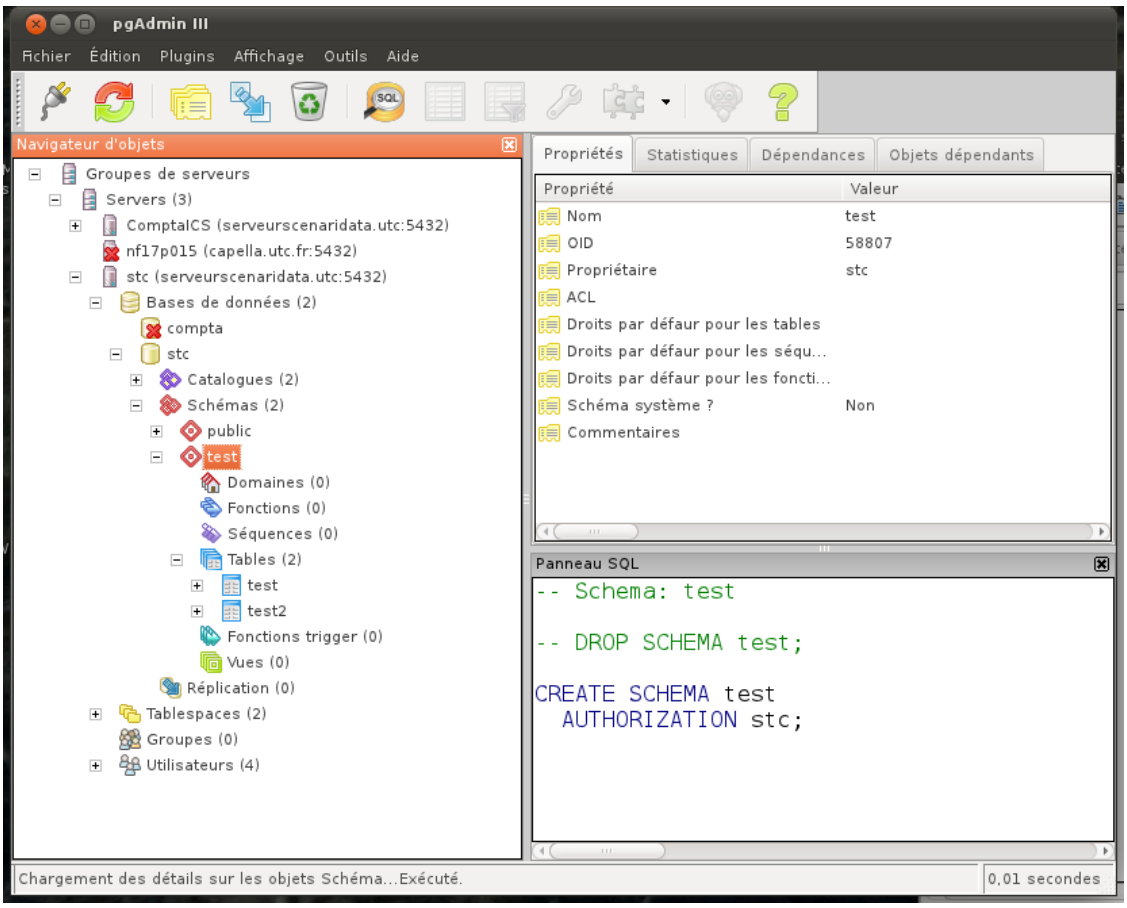


Schéma sous PostgreSQL

Catalogue : schémas

§ *Syntaxe*

`\dn` : Liste des schémas de ma base de données

Schéma par défaut : search_path

Complément

Afin d'alléger la syntaxe il est possible de définir un schéma par défaut, dans lequel seront créer les tables non-préfixées et un ou plusieurs schémas par défaut dans lesquels seront requêtées les tables non-préfixées.

```
SET search_path TO myschema,public;
```

Cette instruction définit le schéma `myschema` comme schéma par défaut pour la création de table et le requêtage, puis `public` pour le requêtage, le premier étant prioritaire sur le second :

- `CREATE mytable` créera ma `mytable` dans le schéma `mychema`.
- `SELECT FROM mytable` cherchera la table dans la schéma `mychema`, puis dans le schéma `public` si la table n'existe pas dans le premier schéma.

Schéma "public" *Remarque*

Le schéma `public` est un schéma créé par défaut à l'initialisation de la base, et qui sert de schéma par défaut en l'absence de toute autre spécification.

Catalogue : \d *Remarque*

La liste des tables retournée par `\d` dépend du *search_path*. Pour que `\d` retourne les tables de `schema1`, il faut donc exécuter l'instruction :

```
SET search_path TO public, schema1
```

Pour aller plus loin *Complément*

http://www.postgresql.org/docs/8.4/static/ddl-schemas.html

2.5. PostgreSQL sous Linux

Exécuter une instruction PostgreSQL depuis Linux *Syntaxe*

```
psql -c "instruction psql"
```

```
psql -h localhost -U user -d db -c "instruction psql"
```

Exécuter un script PostgreSQL depuis Linux *Exemple*

```
#!/bin/sh
psql -c "DROP DATABASE mydb"
psql -c "CREATE DATABASE mydb"
psql -c "GRANT ALL PRIVILEGES ON DATABASE mydb TO user1"
```

psql : exécuter une commande Linux (Linux sous Postgres) *Complément*

- `\!` : permet d'exécuter certaines commandes du *shell* Linux depuis le client *psql*.

2.6. Les clients graphiques pgAdminIII et phpPgAdmin

pgAdminIII

Un client graphique une interface graphique permettant d'effectuer les mêmes opérations qu'avec le client `psql`.

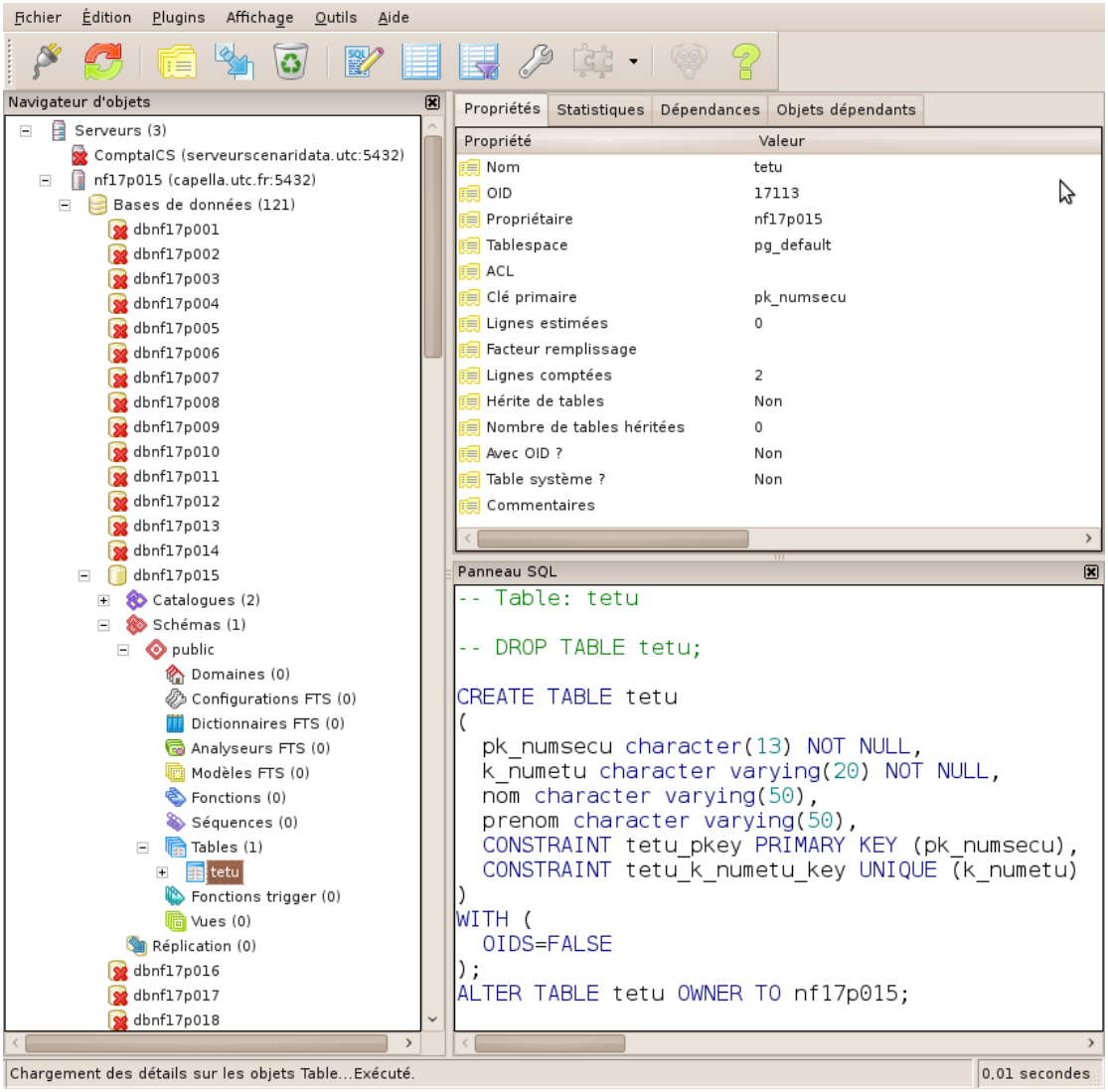
- Le client graphique pgAdminIII est un client lourd qui fonctionne très bien sous Linux et sous Windows.
- Le client graphique phpPgAdmin est un client léger (qui tourne dans un navigateur Web donc).

Déclarer une connexion dans pgAdminIII

1. Sélectionner `Fichier > Ajouter un serveur`
2. Saisissez les informations de connexion à un serveur PostgreSQL existant :
   - Hôte : tuxa.sme.utc
   - Port : 5432 (port standard de PostgreSQL)
   - Base : dbnf17p...
   - Nom : nf17p...
   - Mot de passe : ...

Ouvrir un terminal SQL dans pgAdminIII

1. Sélectionner sa base de données dans la liste `Bases de données`
2. Sélectionner `Outils > Éditeur de requêtes` (ou `CTRL+E`)



pgAdminIII

phpPgAdmin

Complément

http://phppgadmin.sourceforge.net

2.7. Compléments

Héritage (clause INHERITS) *Complément*

http://www.postgresql.org/docs/current/static/sql-createtable.html

PL/pgSQL *Complément*

http://www.postgresql.org/docs/current/static/plpgsql.html

Autres langages procéduraux (PL) *Complément*

- PL/Tcl
- PL/Perl
- PL/Python
- ...

http://www.postgresql.org/docs/current/static/xplang.html
http://www.postgresql.org/docs/current/static/server-programming.html

Triggers *Complément*

http://www.postgresql.org/docs/current/static/sql-createtrigger.html
(http://www.postgresql.org/docs/current/static/triggers.html)

Exercices

1. Découverte d'un SGBDR avec PostgreSQL

1.1. Configuration technique pour réaliser les exercices

Serveur Linux + Client Linux *Fondamental*

La meilleure configuration pour réaliser ces exercices est de disposer :

1. d'un serveur Linux hébergeant un serveur PostgreSQL ;
2. d'une base de données déjà créée (cela peut être la base par défaut *postgres*) ;
3. d'un utilisateur ayant les pleins droits sur cette base de données (cela peut être l'utilisateur par défaut *postgres*) ;
4. d'un client *psql* sous Linux connecté à la base de données.

Serveur Linux + Client Windows

Si un serveur est disponible sous Linux mais que le client est sous Windows, il y a deux solutions :

- **Se connecter en SSH au serveur Linux avec *Putty*∗ (on est ramené au cas précédent).**
- Installer le client *psql* sous Windows, mais cette solution est déconseillée :
  - le terminal est bien moins confortable que sous Linux,
  - une partie des questions sont relatives aux systèmes Linux et ne pourra être réalisée.

Serveur Windows + Client Windows *Complément*

Cette configuration permettra de faire une partie des exercices, avec des adaptations.

1.2. Exercice : Connexion à une base de données PostgreSQL

Cet exercice commence alors que vous êtes connecté à une base Oracle avec le client *psql*.

```
psql (9.x.x)
Saisissez « help » pour l'aide.

mydb=>
```

Demander de l'aide

Demander de l'aide en testant :

- help
- \?
- \h
- \h CREATE TABLE
- \h SELECT

1.3. Exercice : Créer une base de données avec PostgreSQL

Créer une table

Exécuter les instructions suivantes.

```
CREATE TABLE etu (
pknumsecu CHAR(13) PRIMARY KEY,
knumetu VARCHAR(20) UNIQUE NOT NULL,
nom VARCHAR(50),
prenom VARCHAR(50));
```

```
INSERT INTO etu (pknumsecu, knumetu, nom, prenom)
VALUES ('1800675001066', 'AB3937098X', 'Dupont', 'Pierre');
INSERT INTO etu (pknumsecu, knumetu, nom, prenom)
VALUES ('2820475001124', 'XGB67668', 'Durand', 'Anne');
```

```
CREATE TABLE uv (
pkcode CHAR(4) NOT NULL,
fketu CHAR(13) NOT NULL,
PRIMARY KEY (pkcode, fketu),
FOREIGN KEY (fketu) REFERENCES etu(pknumsecu));
```

```
INSERT INTO uv (pkcode, fketu)
VALUES ('NF17', '1800675001066');
INSERT INTO uv (pkcode, fketu)
VALUES ('NF26', '1800675001066');
INSERT INTO uv (pkcode, fketu)
VALUES ('NF29', '1800675001066');
```

Question 1

Utiliser le catalogue pour vérifier la création de la table.

Question 2

Utiliser deux instructions SELECT pour vérifier le contenu de la table.

1.4. Exercice : Import de données depuis un fichier CSV

Nous allons à présent réinitialiser la base avec des données contenues dans un fichier.

Question 1

Exécuter les instructions nécessaires afin de **supprimer** les données existantes dans les tables (instruction DELETE du SQL LMD). Vérifier que les tables sont vides.

Question 2

Créer les deux fichiers de données suivants, respectivement *etus.csv* et *uvs.csv*. Regarder le contenu des fichiers. Pourquoi les appelle-t-on des fichiers CSV ?

```
1;A;Dupont;Pierre
2;B;Durand;Georges
3;C;Duchemin;Paul
4;D;Dugenou;Alain
5;E;Dupied;Albert
```

```
1;NF17
1;NF18
1;NF19
1;NF20
1;LA13
1;PH01
2;NF17
2;NF18
2;NF19
2;TN01
2;LA14
```

```
2;PH01
3;NF17
3;NF18
3;NF19
3;NF21
3;LA14
3;PH01
4;NF17
4;NF20
4;NF21
4;GE10
4;LA14
4;PH01
5;NF17
5;NF18
5;NF20
5;GE10
5;PH01
5;PH02
```

Indice :

Fichier CSV *(cf. p.24)*

Question 3

Insérer les données en complétant les instructions suivantes.

```
\copy ... (...) FROM '...' WITH CSV DELIMITER '...'
\copy ... (...) FROM '...' WITH CSV DELIMITER '...'
```

Indice :

Importer un fichier CSV *(cf. p.8)*

Question 4

Écrivez les requêtes permettant d'obtenir le nombre d'UV suivies par un étudiant et le nombre d'étudiants inscrits par UV.

1.5. Exercice : Manipulation de schémas

Question 1

Visualiser la liste des schémas existants dans votre base de données.

Indice :

Notion de schéma *(cf. p.9)*

Question 2

Créer un second schéma *loisir*.

Question 3

Créer une table *sport* dans le schéma *loisir* ; l'objectif est d'indiquer pour chaque étudiant, la liste des sports qu'il pratique, par exemple : Tennis, Karaté, Aviron...

Vérifier votre création dans le dictionnaire.

1.6. Exercice : Exécution de fichiers SQL et de scripts Linux

Reporter dans un fichier SQL l'ensemble des instruction permettant de supprimer, de créer et d'instancier les tables de la base de données.

Question 1

Exécuter ce fichier depuis *psql*.

Indice :

Exécuter des instructions SQL depuis un fichier *(cf. p.7)*

Question 2

Exécuter ce fichier depuis un script Linux.

Indice :

PostgreSQL sous Linux *(cf. p.11)*

1.7. Exercice : Test de pgAdminIII

Connexion depuis un PC avec pgAdmin III

Installer si nécessaire, puis lancer et tester le programme `pgAdminIII`.

Exécuter une ou deux requêtes permettant de se familiariser avec son fonctionnement.

Complément

1. Installation de PostgreSQL et connexion client-serveur

1.1. Installation de PostgreSQL

Attention

Nous présentons ici une installation *a minima* de PostgreSQL **uniquement à des fins de test et d'apprentissage**, et pour un usage local. Pour mettre en production une base de données PostgreSQL sur le réseau, il faut suivre des directives supplémentaires, notamment pour assurer la sécurité du système, sa sauvegarde...

Ces thèmes ne sont pas abordés dans le cadre de ce cours.

PostgreSQL est à l'origine une base de données conçue pour Unix, son installation et son fonctionnement sont possibles aujourd'hui sur plusieurs OS, mais Linux reste son environnement de prédilection. C'est l'architecture PostgreSQL sur Linux qui est étudiée ici.

Fondamental

https://www.postgresql.org/download/

Installation sous Debian ou Ubuntu à partir des paquets de la distribution

Exemple

`apt install postgresql`

- Ce méta-package permet d'installer le serveur *postgresql* et le client *psql* (ainsi que le client graphique *pgAdmin*).
- Il lance le service *postgresql* (que l'on peut voir en tant que process avec `ps aux | grep postgres` et en tant que service avec `systemctl status postgresql.service`).
- Il crée un utilisateur Linux *postgres* (que l'on peut voir dans `/etc/passwd`).
- Il crée également une base de données par défaut nommée *postgres* et un utilisateur par défaut pour le SGBD nommé *postgres*.

Tester son installation

Méthode

```
sudo su postgres
psql
```

- `sudo su postgres` permet de devenir l'utilisateur Linux *postgres* qui a été créé par l'installation
- `psql` permet d'établir la connexion par défaut entre le client *psql* et le serveur *postgresql* en utilisant l'utilisateur Linux et une base de données du même nom, donc ici *postgres*.

- Le client connecté affiche son numéro de version, propose de taper *help* et une invite de commande `postgres=#`.
- `\l` permet d'afficher la liste des base de données existantes (on retrouve la base *postgres* à laquelle on est connecté).
- `\q` permet de quitter.

Complément

https://doc.ubuntu-fr.org/postgresql

Installer PostgreSQL sous Windows *Complément*

1. Télécharger un *installer* depuis https://www.postgresql.org/download/windows[1]
2. Exécuter l'installation en validant les propositions par défaut (et sans installer les éventuels programmes complémentaires proposés à l'issue de l'installation)
3. Exécuter le client *psql* (également appelé *Shell SQL*)

SHOW ALL *Complément*

La commande `show all` permet de voir tous les paramètres du serveur PostgreSQL.

Par exemple `data_directory` permet de connaître le répertoire de stockage utilisé sur le disque dur.

1.2. Principes de connexion à PostgreSQL

Fondamental

La connexion sur un serveur PostgreSQL nécessite de lui communiquer :

- le nom de la machine et le port sur lesquels tourne le PostgreSQL,
- le nom d'un utilisateur existant sur le PostgreSQL (éventuellement un mot de passe),
- le nom d'une base de données existante dans le PostgreSQL.

Base de données par défaut *Rappel*

L'installation crée une base de données par défaut qui s'appelle *postgres* et un utilisateur *postgres* qui possède cette base (`OWNER`).

Attention

Il ne faut pas confondre le compte utilisateur Linux et le compte utilisateur PostgreSQL car, même s'ils ont le même nom, ce sont deux entités différentes sans lien obligatoire.

Il y a donc :

- un utilisateur *postgres* pour Linux
- un serveur de base de données *postgresql*
- un utilisateur *postgres* pour PostgreSQL
- une base de données *postgres* pour PostgreSQL

[1] https://www.postgresql.org/download/windows/

Superuser *Fondamental*

L'utilisateur *postgres* est un *superuser*, il peut tout faire sur le serveur PostgreSQL, en particulier créer les nouveaux utilisateurs et les nouvelles base de données.

Complément

Dès lors que vous accéderez au serveur PostgreSQL, l'utilisateur connecté va déterminer les droits obtenus vis-à-vis de chacun des objets de la base de données.

Informations réseau *Complément*

- Le port standard de communication de PosgreSQL est *5432*.
- Si le client et le serveur sont sur la même machine, le client peut se connecter au serveur *localhost*.

1.3. Connexion par la méthode "peer" à PostgreSQL

La méthode d'authentification "peer"

La méthode "peer" est utilisée pour se connecter localement à la base de données depuis le client *psql*.

La méthode "peer" utilise le nom d'utilisateur dans le système d'exploitation et l'applique comme nom d'utilisateur de la base de données, sans demander de mot de passe.

Syntaxe

```
1 psql -d database
```

Connexion locale (méthode "peer") *Exemple*

```
1 stc@supermachine:~$ psql -d postgres
```

La commande *psql* lancée par l'utilisateur *stc* avec le paramètre *-d postgres* essaye de se connecter au serveur :

- sur la machine locale (aucune machine n'est spécifiée)
- sur le port 5432 (le port par défaut de *postgresql*)
- avec un utilisateur PostgreSQL nommé *stc* (qui a le même nom que l'utilisateur Linux),
- sans mot de passe (PostgreSQL s'en remet en fait à l'authentification Linux, on parle de méthode d'authentification *peer*),
- à la base de données *postgres* (spécifiée par le paramètre *-d*)

Pour que cette commande fonctionne, il faut que l'utilisateur *stc* et la base de données *postgres* existe dans PostgreSQL.

Créer son utilisateur PostgreSQL *Méthode*

On pose que votre compte Linux s'appelle *alan*, pour vous permettre de vous connecter à PostgreSQL, il faut créer un utilisateur *alan* sur PostgreSQL.

1. Se connecter au serveur en tant que *postgres* avec la méthode "peer"
   - `sudo su postgres`
   - `psql`

2. Une fois connecté, créer un utilisateur *alan* : `CREATE USER alan ;`
3. Se déconnecter de psql (`\q`) puis de l'utilisateur Linux *postgres* (`exit`).
4. Se connecter en tant que *alan* à la base de données *postgres* : `psql -d postgres`

1.4. Connexion par mot de passe à PostgreSQL

La méthode d'authentification "peer"

La méthode de connexion par mot de passe permet de se connecter via le réseau en transmettant un mot de passe chiffré entre le client et le serveur.

Syntaxe

```
psql -h host -U user -d database
```

Connexion distante (méthode par mot de passe) *Exemple*

```
stc@supermachine:~$ psql -h localhost -U alan -d postgres
```

Le client *psql* se connecte sur la machine locale (au port standard) avec l'utilisateur *alan* à la base de données *postgres*.

Remarque

Ici l'utilisateur Linux est *stc*, mais l'utilisateur PostgreSQL est *alan*.

Il n'y a pas avec cette méthode de lien entre l'utilisateur Linux et l'utilisateur PostgreSQL.

Attention

Pour qu'une connexion par mot de passe soit possible, il est nécessaire que l'utilisateur ait un mot de passe au sein de PostgreSQL.

Attribuer un mot de passe à l'utilisateur Postgres *Méthode*

1. Se connecter au serveur en tant que *postgres* avec la méthode "peer"
   - `sudo su postgres`
   - `psql`
2. Une fois connecté, attribuez un mot de passe à l'utilisateur concerné (cela peut-être *postgres* lui même) : `ALTER USER postgres PASSWORD 'test';`
3. Se déconnecter de psql (`\q`)
4. Se connecter en utilisant la méthode par mot de passe : `psql -h localhost -U postgres -d postgres`
5. Entrez le mot de passe

Complément

Il existe trois méthodes majeures (*scram-sha-256*, *md5*, *password*) pour ce type d'authentification. On utilise par défaut la méthode *md5*.

https://docs.postgresql.fr/current/auth-password.html

1.5. Créer des bases de données et des utilisateurs

Créer un utilisateur — *Syntaxe*

```
1 CREATE USER user1 PASSWORD 'password';
```

Créer une base de données — *Syntaxe*

```
1 CREATE DATABASE mydb OWNER user1;
```

La clause OWNER permet de spécifier le propriétaire (*owner*) de la base de données. Celui-ci a tous les droits sur sa base de données. Il pourra créer, modifier et détruire les tables de le base de données.

Supprimer des bases de données et des utilisateurs — *Complément*

```
1 DROP DATABASE mydb;
2 DROP USER user1;
```

Modifier le mot de passe d'un utilisateur — *Complément*

```
1 ALTER USER user1 PASSWORD 'mypassword'
```

Changer le propriétaire d'une base de données — *Complément*

```
1 ALTER DATABASE mydb OWNER TO user2;
```

Catalogue : utilisateurs et bases de données — *Syntaxe*

- `\du` : liste des utilisateurs
- `\l` : liste des bases de données

psql : changer d'utilisateur — *Syntaxe*

`\c db user` : se connecter à la base *db* avec le compte *user*.

1.6. Serveur postgresql

Gestion du serveur

Le serveur s'appelle *postgresql* et se gère avec *systemctl*.

```
1 systemctl stop postgresql
2 systemctl start postgresql
3 systemctl status postgresql
```

Configuration

La configuration du serveur se trouve ici : `/etc/postgresql/X.X/main`

2. Exercice : Alan et Enigma

L'objectif de cet exercice est se créer un utilisateur Alan et une base de données Enigma, et de donner accès par mot de passer à Alan à Enigma.

Cet exercice suppose l'accès à une machine Linux avec les droits administrateur. Procéder à une installation de PostgreSQL le cas échéant.

Question 1

Connectez-vous en tant que *superuser* à PostgreSQL.

Indice :

Pour pouvoir faire nos manipulations, il est nécessaire d'avoir les droits administrateur PostgreSQL. Il faut donc se connecter en tant qu'utilisateur *postgres*.

Question 2

Créez un utilisateur *alan*.

Indice :

Comme il devra se connecter par mot de passe, on le crée tout de suite avec un mot de passe.

Question 3

Créez une base de données *enigma*.

Indice :

Comme elle devra être accédée par *alan*, on lui en donne tout de suite la propriété.

Question 4

Vérifiez les bases de données et utilisateurs existants.

Indice :

Créer des bases de données et des utilisateurs *(cf. p.22)*

Question 5

Connectez-vous en tant qu'*alan* à *enigma*.

Indice :

On ressort de *psql* avec \q, puis on entre à nouveau.

Question 6

Créez une table *bombe* avec deux champs entiers *x* et *y*.

Indice :

```
1 CREATE TABLE ... (
2 ... INTEGER,
3 ... INTEGER
4 )
```

Question 7

Insérez des valeurs dans la table *bombe*.

Indice :

```
1 INSERT INTO ... VALUES (0,0)
```

Question 8

Vérifiez les informations entrées.

Indice :

```
1 SELECT * FROM ...
```

Contenus annexes

1. Fichier CSV

Fichier CSV — *Définition*

CSV∗ est un format informatique permettant de stocker des données tabulaires dans un fichier texte.

Chaque ligne du fichier correspond à une ligne du tableau. Les valeurs de chaque colonne du tableau sont séparées par un caractère de séparation, en général une **virgule** ou un **point-virgule**. Chaque ligne est terminée par un **caractère de fin de ligne** (*line break*).

Toutes les lignes contiennent **obligatoirement** le même nombre de valeurs (donc le même nombre de caractères de séparation). Les valeurs vides doivent être exprimées par deux caractères de séparation contigus.

La taille du tableau est le nombre de lignes multiplié par le nombre de valeurs dans une ligne.

La première ligne du fichier peut être utilisée pour exprimer le nom des colonnes.

Syntaxe

```
[NomColonne1;NomColonne2;...;NomColonneN]
ValeurColonne1;ValeurColonne2;...;ValeurColonneN
ValeurColonne1;ValeurColonne2;...;ValeurColonneN
...
```

Fichier CSV sans entête — *Exemple*

```
Pierre;Dupont;20;UTC;NF17
Pierre;Dupont;20;UTC;NF26
Paul;Durand;21;UTC;NF17
Jacques;Dumoulin;21;UTC;NF29
```

Fichier CSV avec entête — *Exemple*

```
Prenom;Nom;Age;Ecole;UV
Pierre;Dupont;20;UTC;NF17
Pierre;Dupont;20;UTC;NF26
Paul;Durand;21;UTC;NF17
Jacques;Dumoulin;21;UTC;NF29
```

Valeur nulle — *Exemple*

```
Jacques;Dumoulin;;UTC;NF29
```

L'âge est inconnu (NULL).

Variations...

Attention

La syntaxe des fichiers CSV n'est pas complètement standardisée, aussi des variations peuvent exister :

- Les chaînes de caractères peuvent être protégées par des guillemets (les guillemets s'expriment alors avec un double guillemet).
- Le caractère de séparation des nombres décimaux peut être le point ou la virgule (si c'est la virgule, le caractère de séparation doit être différent)
- ...

Un des problème les plus importants reste l'encodage des caractères qui n'est pas spécifié dans le fichier et peut donc être source de problèmes, lors de changement d'*OS*∗ typiquement.

Usage en base de données

Méthode

Les fichiers CSV sont très utilisés en *BD*∗ pour échanger les données d'une table (export/import).

Les *SGBD*∗ contiennent généralement des utilitaires permettant d'exporter une table ou un résultat de requête sous la forme d'un fichier CSV, en spécifiant un certain nombre de paramètres (caractère de séparation de valeur, caractère de fin de ligne, présence ou non d'une ligne de définition des noms des colonnes, etc.). De même ils proposent des utilitaires permettant d'importer un fichier CSV dans une table (en spécifiant les mêmes paramètres), voire de créer directement une table à partir du fichier CSV (quand les noms des colonnes sont présents).

Fichiers à largeur de colonne fixe

Complément

Les fichiers à largeur de colonne fixe n'utilisent pas de séparateur de colonne, mais imposent **le même nombre de caractères** pour chaque cellule. L'avantage est de ne pas avoir à spécifier le caractère de séparation, l'inconvénient est la taille de fichier supérieure si les valeurs ne font pas toutes la même largeur.

XML

Complément

Les fichiers *XML*∗ tendent de plus en plus à remplacer les fichiers CSV car ils permettent d'être beaucoup plus expressifs sur le schéma d'origine. Ils sont également plus standards (encodage spécifié, principe de séparation des données par les *tags*, etc.). Leur seul inconvénient est d'être plus verbeux et donc plus volumineux.

Tables externes

Complément

Certains SGBD, comme Oracle, permettent de créer des tables dites **externes**, qui autorisent de créer un schéma de table **directement sur un fichier CSV**, permettant ainsi un accès SQL standard à un fichier CSV, sans nécessité de l'importer d'abord dans une table.

Glossaire

Client

Un client est un programme informatique qui a pour fonction d'envoyer des requêtes à un autre programme informatique, appelé serveur, d'attendre le résultat de cette requête et de traiter le résultat de la requête. Notons qu'un programme peut-être client vis à vis d'un programme et serveur vis à vis d'un autre. On ne prend pas ici le terme client dans son acception matérielle, qui signifie alors un ordinateur qui a pour fonction d'héberger des programmes clients.

Serveur

Un serveur est un programme informatique qui a pour fonction de recevoir des requêtes d'un autre programme, appelé client, de traiter ces requêtes et de renvoyer en retour une réponse. Notons qu'un programme peut-être serveur vis à vis d'un programme et client vis à vis d'un autre. On ne prend pas ici le terme serveur dans son acception matérielle, qui signifie alors un ordinateur qui a pour fonction d'héberger des programmes serveurs.

Abréviations

BD : Base de Données
CSV : Comma Separated Values
OS : Operating Système (Système d'Exploitation)
SGBD : Système de Gestion de Bases de Données
XML : eXtensible Markup Language

Références

Putty

PuTTy est utile si vous êtes sur un système Windows, pour obtenir une connexion SSH et une console sur un serveur Linux.

C'est un exécutable qui ne nécessite pas d'installation, il peut donc être téléchargé sur n'importe quel disque et lancé directement.

http://www.putty.org/

http://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html