PostgreSQL/PostGIS sous Debian Testing

INDEXATION DU DOCUMENT

	TITRE : PostgreSQL/PostGIS sous Debian Testing	*REFERENCE* :

ACTION	*NOM*	*DATE*	*SIGNATURE*
RÉDIGÉ PAR	Jean David TECHER	2005-04-22	

SUIVI DU DOCUMENT

INDICE	DATE	MODIFICATIONS	NOM

Table des matières

Table des figures

Résumé

Ce guide fournit une note d'installation de PostgreSQL/PostGIS sous GNU/Linux pour Debian Testing. J'ai ici tenté de fournir un guide assez détaillé et complet sur les diverses étapes qui permettent de mettre en place un SGDBR comme PostgreSQL digne de ce nom.

Chapitre 1

Compilation et Installation sur GNU/Linux pour Debian Testing à partir des sources

Nous allons ici procéder aux diverses compilations et installations. Les diverses étapes doivent avoir lieu dans l'ordre chronologique des sections successives de ce chapitre. Nous commencerons par définir une hiérarchie des répertoires des sources pour pouvoir mieux nous retrouver pour la suite et accueillier les sources que nous aurons téléchargées.

Le dossier de téléchargement pour les sources auront lieu vers le répertoire-racine /mnt/sources/

Nous allons lancer un terminal. Nous profiterons aussi lors de ce chapitre pour fournir les commandes permettant d'installer Geos, Proj et PostGIS

NOTE
Je pars ici du prinicpe que les commandes `apt-get` et `apt-cache` de Debian vous sont familiés. Je ne ferais pas ici usage des outils tel que `synaptics` et `aptitude`

1.1 Création des répertoires des sources

Tapez la commande suivante

```
mkdir /mnt/sources
```

Pour simplifer la création des répertoires précédents, on peut aussi utiliser une boucle FOR :

```
for i in PostgreSQL PostGIS Geos Proj;do mkdir -p /mnt/sources/$i;done
```

1.2 PostgreSQL

1.2.1 Création du super-utilisateur de PostgreSQL

Les programmeurs de PostgreSQL pour des raisons de sécurité (impliquant des failles/fuites de sécurités selon les droits accordés à cet utilisateur sous Linux qui peuvent être dûe à des attaques de requêtes de type "SQL injection" par le réseau ou par une mauvaise utilisation des modules objets chargés dans les bases par de simples utilisateurs etc...) ont imposés que cet utilisateur devait avoir des droits limités sur la machine sur lequel tourne PostgreSQL.Cellà a toutjours été le cas sous des systèmes autres que Windows. Cette règle fut ensuite pris en compte - pour Windows -lors du développement de la 7.5 devel juste avant la sortie de la 8.0.0.

Créons donc cet utilisateur qui a l'accoutumée et pour des raison d'habitude est appelé **postgres** .

NOTE
Vous pouvez appeler cet utilisateur comme bon vous semblera. Veuillez alors adapter la suite de la documentation à votre choix.

```
adduser postgres
```

qui nous renvoit

```
Ajout de l'utilisateur « postgres »...
Ajout du nouveau groupe « postgres » (1002).
Ajout du nouvel utilisateur « postgres » (1002) avec le groupe « postgres ».
Création du répertoire personnel « /home/postgres ».
Copie des fichiers depuis « /etc/skel »
Enter new UNIX password:
Retype new UNIX password:
passwd : le mot de passe a été mis à jour avec succès
Modification des informations relatives à l'utilisateur postgres
Entrez la nouvelle valeur ou « Entrée » pour conserver la valeur proposée
        Nom complet []:
        N° de bureau []:
        Téléphone professionnel []:
        Téléphone personnel []:
        Autre []:
Ces informations sont-elles correctes [o/N] ? o
```

Dans la suite, je ne propose pas de passer en revue les diverses options de compilation de PostgreSQL mais de retenir celles que j'ai l'habitude d'utiliser.

1.2.2 Pré-requis

Ce qui est bon à savoir avant de commencer à compiler PostgreSQL.

Sur le système, il faut commencer par vérifier la présences des outils suivants notamment dans le cadre d'un environnement de compilation complet

make : version supérieure à la 3.7 que l'on peut vérifier en faisant make -v ;

GCC : compilateur C de GNU ;

tar utilitaire de décompression pour les sources de PostgreSQL ;

libreadline bibliothèquequi permet d'avoir accès à l'historique de requête dans le moniteur interactif psql ;

zlib bibiothèque nécessaire pour la création d'archives lors de la sauvegardes de bases de données en archive tar ou comrpessées par pg_dum ou pg_restore ;

gettext Le package Gettext est utilisé pour l'internationalisation et la localisation. Les programmes peuvent êtres compilés avec le support de la langue native ('Native Language Support' ou NLS) qui leur permettent d'afficher les messages dans la langue native de l'utilisateur, notamment pour nous le français.

Par exemple sous debian, on pourrait faire

```
apt-cache search tar
```

et ainsi de suite pour les autres bibliothèques. Puis il faudrait installer l'ensemble manquant en fonction de ce qu'affiche la recherche par la ligne

```
apt-get update
apt-get install tar zlib1g zlib1g-dev libreadline5 libreadline5-dev gettext gettext-base
```

SI l'on souhaite compiler PostgreSQL avec des options spéciales, il est bon d'avoir d'installé

OpenSSL : pour chiffrer et sécuriser les connexions entre l'application client (par exemple psql ou pgadmin) et le serveur

Tcl/Tk indispensable pour utiliser PgAccess (application d'administration graphique pour PostgreSQL)

1.2.3 Téléchargement et décompression

Téléchargez les sources de PostgreSQL version 8.1.1 à cette URL

ftp ://ftp.fr.postgresql.org/source/v8.1.1/postgresql-8.1.1.tar.gz

et copiez ce fichier vers /mnt/sources/PostgreSQL ou plus simplement en faisant

```
cd /mnt/sources/PostgreSQL
wget ftp://ftp.fr.postgresql.org/source/v8.1.1/postgresql-8.1.1.tar.gz
```

Pour la décompression, il nous suffira de faire

```
cd /mnt/sources/PostgreSQL
tar xvzf postgresql-8.1.1.tar.gz
```

1.2.4 Configuration

Cette section va nous permettre de configurer les sources avant leur compilation, nous allons faire appel à un script qui va également effectuer une vérification des dépendances logicielles. Voici la commande nécessaire pour utiliser ce script :

```
./configure
```

La commande suivante nous donnera des informations bien utiles sur les diverses options possibles de compilation

```
./configure --help
'configure' configures PostgreSQL 8.1.1 to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE.  See below for descriptions of some of the useful variables.

Defaults for the options are specified in brackets.

Configuration:
  -h, --help              display this help and exit
      --help=short        display options specific to this package
      --help=recursive    display the short help of all the included packages
  -V, --version           display version information and exit
  -q, --quiet, --silent   do not print 'checking...' messages
      --cache-file=FILE   cache test results in FILE [disabled]
  -C, --config-cache      alias for '--cache-file=config.cache'
  -n, --no-create         do not create output files
      --srcdir=DIR        find the sources in DIR [configure dir or '..']

Installation directories:
  --prefix=PREFIX         install architecture-independent files in PREFIX
                          [/usr/local/pgsql]
  --exec-prefix=EPREFIX   install architecture-dependent files in EPREFIX
                          [PREFIX]

By default, 'make install' will install all the files in
'/usr/local/pgsql/bin', '/usr/local/pgsql/lib' etc.  You can specify
an installation prefix other than '/usr/local/pgsql' using '--prefix',
for instance '--prefix=$HOME'.

For better control, use the options below.

Fine tuning of the installation directories:
  --bindir=DIR           user executables [EPREFIX/bin]
```

```
  --sbindir=DIR           system admin executables [EPREFIX/sbin]
  --libexecdir=DIR        program executables [EPREFIX/libexec]
  --datadir=DIR           read-only architecture-independent data [PREFIX/share]
  --sysconfdir=DIR        read-only single-machine data [PREFIX/etc]
  --sharedstatedir=DIR    modifiable architecture-independent data [PREFIX/com]
  --localstatedir=DIR     modifiable single-machine data [PREFIX/var]
  --libdir=DIR            object code libraries [EPREFIX/lib]
  --includedir=DIR        C header files [PREFIX/include]
  --oldincludedir=DIR     C header files for non-gcc [/usr/include]
  --infodir=DIR           info documentation [PREFIX/info]
  --mandir=DIR            man documentation [PREFIX/man]

System types:
  --build=BUILD     configure for building on BUILD [guessed]
  --host=HOST       cross-compile to build programs to run on HOST [BUILD]

Optional Features:
  --disable-FEATURE       do not include FEATURE (same as --enable-FEATURE=no)
  --enable-FEATURE[=ARG]  include FEATURE [ARG=yes]
  --enable-integer-datetimes  enable 64-bit integer date/time support
  --enable-nls[=LANGUAGES]  enable Native Language Support
  --disable-shared        do not build shared libraries
  --disable-rpath         do not embed shared library search path in executables
  --disable-spinlocks     do not use spinlocks
  --enable-debug          build with debugging symbols (-g)
  --enable-depend         turn on automatic dependency tracking
  --enable-cassert        enable assertion checks (for debugging)
  --enable-thread-safety  make client libraries thread-safe
  --enable-thread-safety-force  force thread-safety in spite of thread test failure
  --disable-largefile     omit support for large files

Optional Packages:
  --with-PACKAGE[=ARG]    use PACKAGE [ARG=yes]
  --without-PACKAGE       do not use PACKAGE (same as --with-PACKAGE=no)

  --with-docdir=DIR       install the documentation in DIR [PREFIX/doc]
  --without-docdir        do not install the documentation
  --with-includes=DIRS    look for additional header files in DIRS
  --with-libraries=DIRS   look for additional libraries in DIRS
  --with-libs=DIRS        alternative spelling of --with-libraries
  --with-pgport=PORTNUM   change default port number 5432
  --with-tcl              build Tcl modules (PL/Tcl)
  --with-tclconfig=DIR    tclConfig.sh is in DIR
  --with-perl             build Perl modules (PL/Perl)
  --with-python           build Python modules (PL/Python)
  --with-krb5             build with Kerberos 5 support
  --with-krb-srvnam=NAME  name of the default service principal in Kerberos [postgres]
  --with-pam              build with PAM support
  --with-bonjour          build with Bonjour support
  --with-openssl          build with OpenSSL support
  --without-readline      do not use Readline
  --without-zlib          do not use Zlib
  --with-gnu-ld           assume the C compiler uses GNU ld [default=no]

Some influential environment variables:
  CC          C compiler command
  CFLAGS      C compiler flags
  LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
              nonstandard directory <lib dir>
  CPPFLAGS    C/C++ preprocessor flags, e.g. -I<include dir> if you have
              headers in a nonstandard directory <include dir>
  CPP         C preprocessor
```

```
  LDFLAGS_SL
  DOCBOOKSTYLE
              location of DocBook stylesheets

Use these variables to override the choices made by 'configure' or to help
it to find libraries and programs with nonstandard names/locations.

Report bugs to <pgsql-bugs@postgresql.org>.
```

Le script `configure` accepte diverses options qui permettent de configurer précisément la façon dont est construit PostgreSQL :

1. **--prefix=PREFIX** : Installe tous les fichiers dans le répertoire PREFIX au lieu de /usr/local/pgsql. Les fichiers seront en fait installés dans divers sous-répertoires ; aucun fichier ne sera installé directement dans le répertoire PREFIX ;
2. **--exec-prefix=EXEC-PREFIX** : Installe les fichiers exécutables dépendants de l'architecture dans le répertoire EXEC-PREFIX. Dans le cas où il n'est pas précisé, EXEC-PREFIX vaut PREFIX ;
3. **--bindir=DIRECTORY** : Indique le répertoire des fichiers exécutables. Par défaut vaut EXEC-PREFIX/bin (soit /usr/local/pgsql/bin par défaut) ;
4. **--datadir=DIRECTORY** : Indique le répertoire des fichiers en lecture seule utilisés par PostgreSQL. Par défaut vaut PREFIX/share. Cela n'a **rien** à voir avec l'emplacement des fichiers de base de données ;
5. **--sysconfdir=DIRECTORY** : Répertoire pour les fichiers de configuration, PREFIX/etc par défaut ;
6. **--libdir=DIRECTORY** : Emplacement des bibliothèques et des modules dynamiques. Par défaut, EXEC-PREFIX/lib ;
7. **--includedir=DIRECTORY** : Répertoire d'installation des fichiers d'en-tête C et C++. Par défaut, PREFIX/include ;
8. **--mandir=DIRECTORY** : Les pages de manuel de PostgreSQL seront installées dans ce répertoire. Par défaut, PREFIX/man ;
9. **--with-docdir=DIRECTORY** : Les fichiers de documentations, sauf les pages de manuel, seront installées dans ce répertoire. Par défaut, PREFIX/doc ;
10. **--without-docdir** : n'installe pas la documentation lors du **make install** ;
11. **--with-includes=DIRECTORIES** : DIRECTORIES est une liste de répertoires séparés par deux-points dans lesquels seront recherchés les fichiers d'include en plus des répertoires standards. Utile si vous avez des bibliothèques (comme GNU Readline) installées dans des répertoires particuliers ;
12. **--with-libraries=DIRECTORIES** : DIRECTORIES est une liste de répertoires séparés par deux-points dans lesquels seront recherchés les bibliothèques. Comme pour **--with-includes**, à n'utiliser que si vous disposez de packages installés dans des emplacements non standards ;
13. **--enable-nls[=LANGUAGES]** : Active l'affichage des messages dans d'autres langues que l'anglais. LANGUAGES est une liste des codes à supporter séparés par des espaces. Si vous n'indiquez rien, toutes les traductions sont installées. Nécessite une implémentation de l'API Gettext ;
14. **--with-pgport=NUMBER** : Paramètre le numéro de port par défaut du serveur et des clients. Par défaut, vaut 5432. A n'utiliser que si vous voulez lancer plusieurs instances de PostgreSQL sur la même machine ;
15. **--with-perl** : Compile le langage PL/Perl coté serveur ;
16. **--with-python** : Compile le langage PL/Python côté serveur ;
17. **--with-tcl** : Compile le langage PL/Tclcôté serveur ;
18. **--with-tclconfig=DIRECTORY** : Chemin contenant les informations de configuration pour l'interfaçage avec Tcl situées dans le fichier tclConfig.sh. A utiliser si vous voulez utiliser une version de Tcl différente de celle installée ;
19. **--with-krb4**, **--with-krb5** : support de l'authentification Kerberos 4 ou 5 mais pas les deux ;
20. **--with-krb-srvnam=NAME** : Nom du service principal de Kerberos ;
21. **--with-openssl** : Support pour les connexions SSL (cryptées). Requiert le package OpenSSL ;
22. **--with-pam** : Support PAM (Pluggable Authentication Modules) ;
23. **--without-readline** : Evite l'utilisation de la bibliothèque Readline ce qui désactive l'historique des commandes et l'édition de la ligne en cours ;
24. **--with-rendezvous** : Support Rendezvous (recommandé sur Mac OS X) ;
25. **--disable-spinlocks** : Permet de compiler PostgreSQL même si aucun support pour le spinlock CPU de la plateforme. Résulte en des performances médiocres ;

26. **--enable-thread-safety** : Rend les bibliothèques client thread-safe ;
27. **--without-zlib** : Evite l'utilisation de la bibliothèque Zlib. Désactive les archives compressées de pg_dump (à n'utiliser que si vous ne disposez pas de zlib) ;
28. **--enable-debug** : Inclut les informations de débuggage dans les binaires et les bilbiothèques, uniquement pour le développement ;
29. **--enable-cassert** : Active les vérifications d'assertions, uniquement pour le développement ;
30. **--enable-depend** ; Active la surveillance automatique des dépendances. Les makefiles résultants forceront la recréation des binaires si un des headers a changé. Ne fonctionne que sur GCC, pour le développement ;
31. Comme pour les scripts configure classiques, vous pouvez paramétrer le compilateur *via* la variable d'environnement CC (par défaut, GCC est utilisé), les drapeaux du compilateur *via* CFLAGS, vous pouvez passer ces variables sur la ligne de commande (./configure CC=/opt/bin/gcc CFLAGS=&#8217 ;-O2 -pipe&#8217 ;).

Nous essayerons de compiler PostgreSQL avec l'options suivante

--enable-nls : permet d'avoir le support de langue adéquate nls (Native Language Support), notamment le français.

1.2.5 Compilation et installation

Si la ligne de configuration s'est passé sans problème alors, on peut passer à la suite, la compilation

```
make
```

Il ne reste plus qu'à installer nos binaires en faisant

```
make install
```

Nous rappelerons au passage que si l'option **--prefix** dans la ligne de **./configure** n'a pas été déteminé par défaut notre distribution sera alors installé par défaut dans le répertoire `/usr/local/pgsql` contenant les sous répertoire `bin`, `include`, `lib`, `doc`, `share`

1.2.6 Ajout des outils/supports contributifs pour PostgreSQL

C'est dans le répertoire `contrib` des sources de PostgreSQL que se trouvent les supports (une bonne partie) offrant plus de fonctionnalités type. Pour pouvoir les ajouter à notre distribution, il nous suffira de faire :

```
cd /mnt/sources/PostgreSQL/postgresql-8.1.1/contrib
make
make install
```

1.2.7 Commandes post-installation

Afin de pouvoir accéder librement à nos utilitaires de PostgreSQL et comme ce dernier dispose de variable d'environnement utiles - que nous verrons en temps utile - comme PGDATA, qui précise le chemin d'accès vers notre groupe de bases de données, nous allons renseigner ces informations en modifiant le fichier `/etc/profile` :

```
PATH=/usr/local/pgsql/bin:$PATH
PGDATA="/mnt/pgdata"
export PATH PGDATA
```

Au lieu de préciser ici la modification à la variable d'environnement LD_LIBRARY_PATH lui demandant de pointer vers le répertoire des librairies de PostgreSQL (`/usr/local/pgsql/lib`), ouvrons le fichier /etc/ld.so.conf et ajoutons la ligne suivante

```
/usr/local/pgsql/lib
```

modification que l'on peut aussi obtenir en faisant

```
echo /usr/local/pgsql/lib >> /etc/ld.so.conf
```

Rechargeons maintenant les librairies dans le "cache" du système de manière à pouvoir les prendre en comptes en faisant

```
ldconfig
```

Si nous sommes toujours dans le même termina ou un autre, rechargeons la variable d'environnement `PATH` du système en faisant

```
source /etc/profile
```

Vérifions que tout semble avoir bien été installé en faisant par exemple

```
if [ ! -z $(which pg_config) ]; then for i in '--version' '--configure' '--includedir';do \
pg_config ${i};done;else echo "PostgreSQL est introuvable";fi
```

qui devrait renvoyer

```
PostgreSQL 8.1.1
'-with-openssl' '--enable-nls'
/usr/local/pgsql/include
```

Personnellement, pour avoir un PATH et un PGDATA accessible par défaut pour tous les utilsiateurs, je mets aussi celà dans le fichier `/etc/bash.bashrc`

```
# System-wide .bashrc file for interactive bash(1) shells.

# To enable the settings / commands in this file for login shells as well,
# this file has to be sourced in /etc/profile.
export PATH="${PATH}:/usr/local/pgsql/bin"
export PGDATA="/mnt/pgdata"
[...]
```

1.2.8 Jeu de caractères Europe occidentales, LATIN9

Pour faire l'initialisation qui sera l'objectif de la prochaine section, il nous faut mettre à jour nos locales. Commencez par exécuter la commande suivante en tant que root `locale -a` et voyez si elle vous renvoit au moins les lignes suivantes

```
[...]
fr_FR
fr_FR@euro
fr_FR.iso88591
fr_FR.iso885915@euro
[...]
```

NOTE
Pour les jeux d'encodage supportés par PostgreSQL 8.1.1, merci de consultez le lien suivant bien utile http ://traduc.postgresqlfr.org/pgsql-8.1.1-fr/multibyte.html qui explique bien les jeux de caractères supportés par le serveur

Si c'est la case vous pouvez passer directement à la section suivante. Sinon il vous faut le jeu de caractères propres à la France, le LATIN9. A cette fin, tapez donc la commande suivante

```
dpkg-reconfigure locales
```

Dans la liste apparaissant sur fond bleu, faites défiler la liste - en utilisant les flèches de votre clavier - et sélectionnez

1. **[*] fr_FR ISO-8859-1**
2. **[*] fr_FR.UTF-8 UTF-8**
3. **[*] fr_FR@euro ISO-8859-15**

NOTE
Pour sélectionner ces options, utilisez la touche [BARRE ESPACE] de votre clavier

Pour sélectionner, le bouton situé en bas appuyez sur la touche [TABULATION] de votre clavier de manière à ce que ce dernier apparaisse sur un fond rouge - voir figure - et appuyez sur [ENTER]

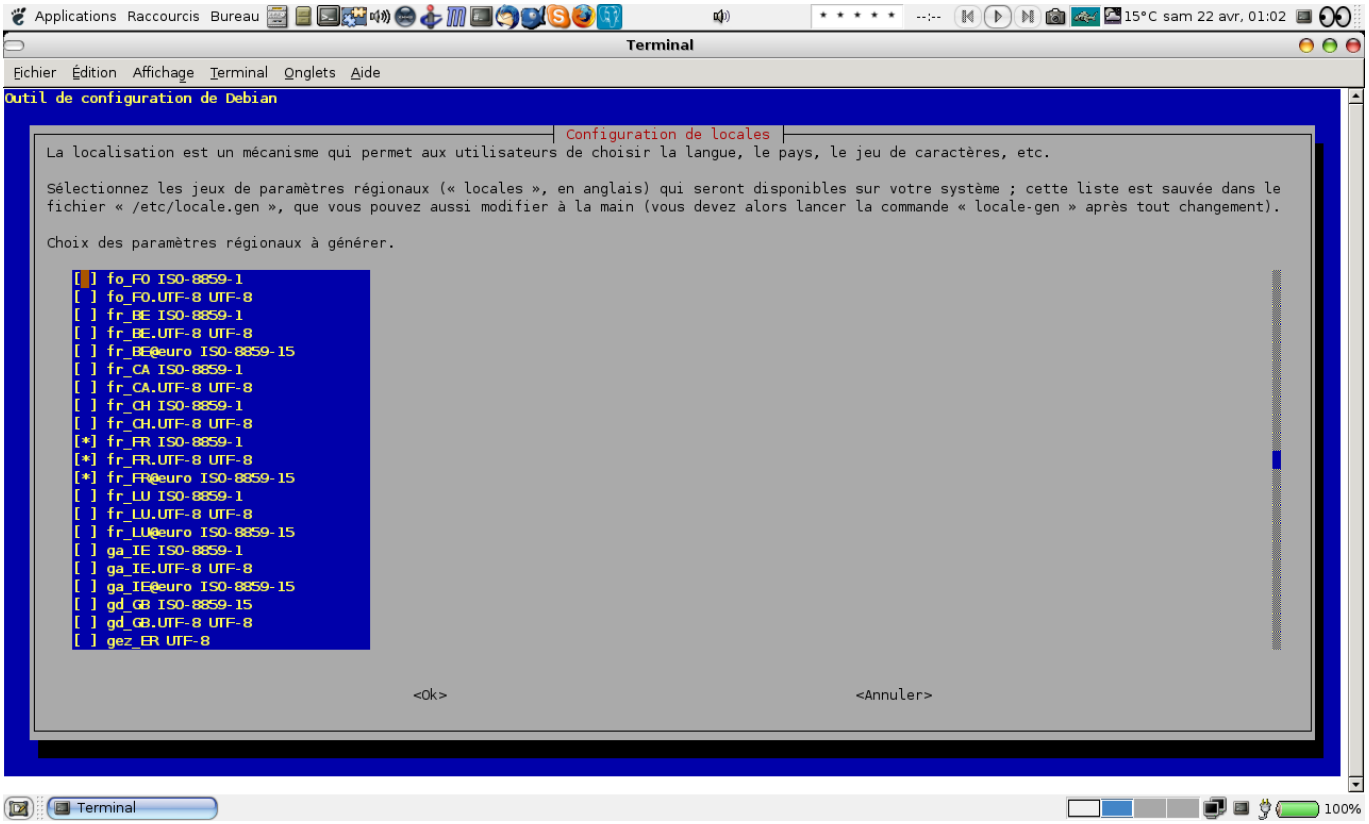

FIG. 1.1 – Configuration des locales : choix des jeux régionaux possibles

Il ne reste pus qu'à préciser le jeu de paramètres régionaux apparaissant sur la nouvelle fenêtre, à savoir fr_FR@euro. Pour vérifier que tout s'est bien passé, il suffit de faire `cat /etc/environment` qui doit renvoyer au moins `LANG=fr_FR@euro`

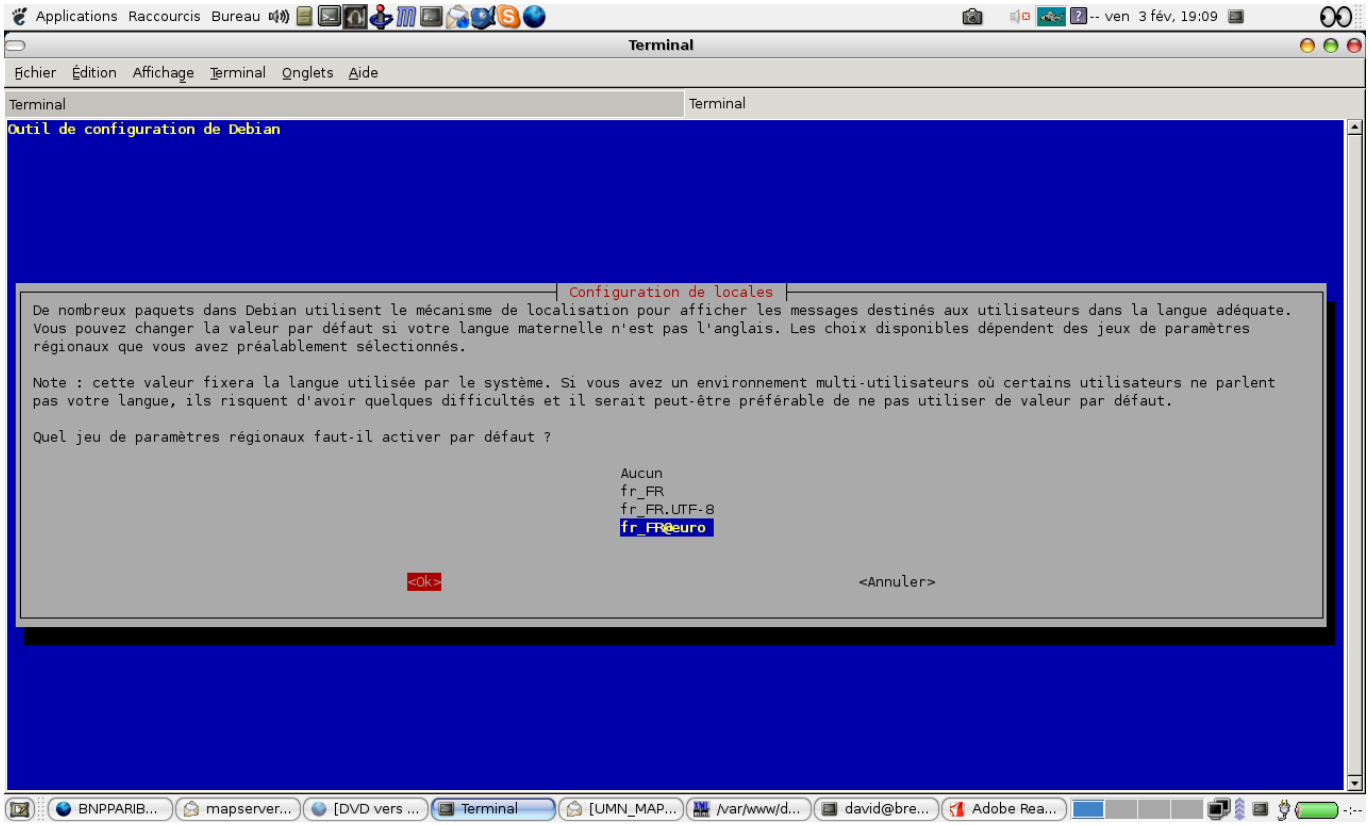

FIG. 1.2 – Configuration de la locale : choix en fr_FR@euro

1.2.9 Initialisation du serveur

L'initialisation - au sens du système de fichiers - consiste à la création du (premier) groupe de vos futures bases de données par défaut - faisant parti d'un cluster au sens des bases de données -. Cela va consister "à remplir" le répertoire correspondant à la variable d'environnement PGDATA de PostgreSQL qui n'existe pas encore. Elle répond aux besoins suivants. Quel encodage choisir par défaut ? Quel doit être le mode de connexion par défaut ? etc...Mais le reste des configurations des paramètrages du serveur peut par la suite être reprise dans les fichiers .conf de PostgreSQL..

C'est l'utilitaire `initdb` qui précise et fait tout celà. Pour obtenir un résumé succint des possibilités offertes par cet utilitaire, il suffit de saisir

```
$initdb --help
initdb initialise un groupe de bases de données PostgreSQL.

Usage :
  initdb [OPTION]... [REP_DONNEES]

Options :
 [-D, --pgdata=]REP_DONNEES emplacement de ce groupe de bases de données
  -E, --encoding=CODAGE     initialise le codage par défaut des nouvelles
                            bases de données
  --locale=LOCALE           initialise le groupe de bases de données avec la
                            locale donnée
  --lc-collate, --lc-ctype, --lc-messages=LOCALE
  --lc-monetary, --lc-numeric, --lc-time=LOCALE
                            initialise le groupe de bases de données avec la
                            locale précisée dans la catégorie respective
                            (prise, par défaut, dans l'environnement)
  --no-locale               équivalent à --locale=C
```

```
  -A, --auth=MÉTHODE        méthode d'authentification par défaut pour les connexions ↩
      locales
  -U, --username=NOM        nom du superutilisateur de la base de données
  -W, --pwprompt            demande un mot de passe pour le nouveau
                            superutilisateur
  --pwfile=nomfichier       lit le mot de passe du nouveau superutilisateur
                            à partir d'un fichier
  -?, --help                affiche cette aide puis quitte
  -V, --version             affiche la version puis quitte

Options moins utilisées :
  -d, --debug               génère un grand nombre de traces de déboguage
  -s, --show                affiche les paramètrages internes
  -L DIRECTORY              indique où trouver les fichiers d'entrées
  -n, --noclean             ne nettoie pas après des erreurs

Si le répertoire des données n'est pas spécifié, la variable d'environnement
PGDATA est utilisée.
```

Créeons maintenant notre répertorie de groupe de bases de données - qui correspond à la variable PGDATA - en faisant

```
mkdir /mnt/pgdata
chown -R postgres:postgres /mnt/pgdata
```

Connectons-nous en tant qu'utilisateur postgres (=super-utilisateur de PostgreSQL) depuis le terminal.

```
su postgres
```

Vérifions malgré tout que le répertoire des utilitaires de PostgreSQL (`/usr/local/pgsql/bin`) soient accessibles en les voyant apparaitre dans notre PATH , depuis PGDATA respectivement

```
env | grep PATH=
```

```
env |grep PGDATA
```

sinon mettons là à jour en faisant

```
source /etc/profile
```

Remplissons maintenant notre répertoire de données selon les spécifications suivantes

Connexions sans mot de passe (mode trusting) sur la machine en locale à tous les utisateurs créés ultérieurement par le super-utilisateur (option -A trust à passer à initdb) ;

encodage selon les locales supportées par la machine

NOTE
Les locales de la machine peuvent être listées en utilisant la commande `locale -a` sur la machine

Ces spécifications sont pris en compte par initdb en lui passant les options suivantes

```
initdb -A trust
```

```
Les fichiers appartenant à ce système de bases de données doivent appartenir à l' ↩
    utilisateur «postgres».
Cet utilisateur doit aussi posséder le processus serveur.

Le groupe de bases de données sera initialisé avec la locale fr_FR@euro.
Le codage de la base de données par défaut a été correctement configuré avec LATIN9.

création du répertoire /mnt/pgdata ... ok
création du répertoire /mnt/pgdata/global... ok
```

```
création du répertoire /mnt/pgdata/pg_xlog... ok
création du répertoire /mnt/pgdata/pg_xlog/archive_status... ok
création du répertoire /mnt/pgdata/pg_clog... ok
création du répertoire /mnt/pgdata/pg_subtrans... ok
création du répertoire /mnt/pgdata/pg_twophase... ok
création du répertoire /mnt/pgdata/pg_multixact/members... ok
création du répertoire /mnt/pgdata/pg_multixact/offsets... ok
création du répertoire /mnt/pgdata/base... ok
création du répertoire /mnt/pgdata/base/1... ok
création du répertoire /mnt/pgdata/pg_tblspc... ok
sélection de la valeur par défaut de max_connections... 100
sélection de la valeur par défaut de shared_buffers... 1000
création des fichiers de configuration... ok
création de la base de données template1 dans /mnt/pgdata/base/1... ok
initialisation de pg_authid... ok
activation de la taille illimitée des lignes pour les tables systèmes... ok
initialisation des dépendances... ok
création des vues système... ok
chargement de pg_description... ok
création des conversions... ok
initialisation des privilèges sur les objets intégrés... ok
création du shéma d'informations... ok
lancement du vacuum sur la base de données template1... ok
copie de template1 vers template0... ok
copie de template1 vers postgres... ok

Succès. Vous pouvez maintenant lancer le serveur de bases de données en utilisant:

    postmaster -D /mnt/pgdata
or
    pg_ctl -D /mnt/pgdata -l journaltrace start
```

Pour les jeux d'encodage supportés par PostgreSQL 8.1.1, merci de consultez le lien suivant bien utile

http ://traduc.postgresqlfr.org/pgsql-8.1.1-fr/multibyte.html qui explique bien les jeux de caractères supportés par le serveur

A savoir, si vous ne souhaitez pas installer PostgreSQL en tant que service mais pouvoir le démarrer/arrêter manuellement, il suffit de faire respectivement

```
pg_ctl start
```

ou

```
pg_ctl stop
```

1.2.10 Installation de PostgreSQL en tant que service

Le script permettant d'installer PostgreSQL en tant que service se trouve à `/mnt/sources/PostgreSQL/postgresql-8.1.1-/contrib/start-scripts/linux` Par exemple sous Debian, on copiera le script `linux` vers `/etc/init.d/postgresql`

```
cp /mnt/sources/PostgreSQL/postgresql-8.1.1/contrib/start-scripts/linux /etc/init.d/ ↩
    postgresql
```

Il est nécessaire de modifier le script de démarrage afin de positionner différentes variables (des commentaires indiquent la zone où il peut être nécessaire d'apporter des modifications). Si vous n'avez rien changé, il n'est pas utile de modifier quoi que ce soit.

Ajoutons les droits nécessaires au fichier

```
chmod 744 /etc/init.d/postgresql
```

Ensuite il faut créer les liens symboliques pour que le service soit démarré aux bons niveaux d'exécution. Sous Debian

```
update-rc.d postgresql defaults
```

Le serveur est maintenant prêt à être démarré. Pour cela, il suffit d'utiliser le script que l'on vient de mettre en place :

```
/etc/init.d/postgresql start
```

A titre d'information voici le contenu de mon fichier /etc/init.d/postgresql

```
#! /bin/sh

# chkconfig: 2345 98 02
# description: PostgreSQL RDBMS

# This is an example of a start/stop script for SysV-style init, such
# as is used on Linux systems.  You should edit some of the variables
# and maybe the 'echo' commands.
#
# Place this file at /etc/init.d/postgresql (or
# /etc/rc.d/init.d/postgresql) and make symlinks to
#   /etc/rc.d/rc0.d/K02postgresql
#   /etc/rc.d/rc1.d/K02postgresql
#   /etc/rc.d/rc2.d/K02postgresql
#   /etc/rc.d/rc3.d/S98postgresql
#   /etc/rc.d/rc4.d/S98postgresql
#   /etc/rc.d/rc5.d/S98postgresql
# Or, if you have chkconfig, simply:
# chkconfig --add postgresql
#
# Proper init scripts on Linux systems normally require setting lock
# and pid files under /var/run as well as reacting to network
# settings, so you should treat this with care.

# Original author:  Ryan Kirkpatrick <pgsql@rkirkpat.net>

# $PostgreSQL: pgsql/contrib/start-scripts/linux,v 1.7 2004/10/01 18:30:21 tgl Exp $

## EDIT FROM HERE

# Installation prefix
prefix=/usr/local/pgsql

# Data directory
PGDATA="/mnt/pgdata"

# Who to run the postmaster as, usually "postgres".  (NOT "root")
PGUSER=postgres

# Where to keep a log file
#PGLOG="/var/log/pgsql"

## STOP EDITING HERE

# Check for echo -n vs echo \c
if echo '\c' | grep -s c >/dev/null 2>&1 ; then
    ECHO_N="echo -n"
    ECHO_C=""
else
    ECHO_N="echo"
    ECHO_C='\c'
fi
```

```
# The path that is to be used for the script
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# What to use to start up the postmaster (we do NOT use pg_ctl for this,
# as it adds no value and can cause the postmaster to misrecognize a stale
# lock file)
DAEMON="$prefix/bin/postmaster"

# What to use to shut down the postmaster
PGCTL="$prefix/bin/pg_ctl"

set -e

# Only start if we can find the postmaster.
test -x $DAEMON || exit 0

# Parse command line parameters.
case $1 in
  start)
  $ECHO_N "Starting PostgreSQL: "$ECHO_C
  su - $PGUSER -c "$DAEMON -D '$PGDATA' &"  2>&1
  echo "ok"
  ;;
  stop)
  echo -n "Stopping PostgreSQL: "
  su - $PGUSER -c "$PGCTL stop -D '$PGDATA' -s -m fast"
  echo "ok"
  ;;
  restart)
  echo -n "Restarting PostgreSQL: "
  su - $PGUSER -c "$PGCTL stop -D '$PGDATA' -s -m fast -w"
  su - $PGUSER -c "$DAEMON -D '$PGDATA' &"  2>&1
  echo "ok"
  ;;
  reload)
        echo -n "Reload PostgreSQL: "
        su - $PGUSER -c "$PGCTL reload -D '$PGDATA' -s"
        echo "ok"
        ;;
  status)
  su - $PGUSER -c "$PGCTL status -D '$PGDATA'"
  ;;
  *)
  # Print help
  echo "Usage: $0 {start|stop|restart|reload|status}" 1>&2
  exit 1
  ;;
esac

exit 0
```

Ce qu'il est possible de faire afin d'avoir une trace de chaque fichier de log - lors du redémarrage de la machine - c'est de spécifier un répertoire de log où seront conservés les fichiers de log. Pour ma part, j'ai opté pour là création d'un sous-répertoire dans $PGDATA = /mnt/pgdata

```
su postgres
mkdir $PGDATA/pg_log
```

J'ai ensuite modifié mon fichier /mnt/pgdata/postgresql.conf de la ligne 203 à 240 dont voici un extrait

```
#---------------------------------------------------------------------------
# ERROR REPORTING AND LOGGING
```

```
#---------------------------------------------------------------------------

# - Where to Log -

#log_destination = 'stderr'   # Valid values are combinations of
          # stderr, syslog and eventlog,
          # depending on platform.

# This is used when logging to stderr:
redirect_stderr = on      # Enable capturing of stderr into log
          # files

# These are only used if redirect_stderr is on:
log_directory = 'pg_log'    # Directory where log files are written
          # Can be absolute or relative to PGDATA
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log' # Log file name pattern.
          # Can include strftime() escapes
log_truncate_on_rotation = off # If on, any existing log file of the same
          # name as the new log file will be
          # truncated rather than appended to. But
          # such truncation only occurs on
          # time-driven rotation, not on restarts
          # or size-driven rotation. Default is
          # off, meaning append to existing files
          # in all cases.
log_rotation_age = 1440   # Automatic rotation of logfiles will
          # happen after so many minutes.  0 to
          # disable.
log_rotation_size = 10240   # Automatic rotation of logfiles will
          # happen after so many kilobytes of log
          # output.  0 to disable.

# These are relevant when logging to syslog:
#syslog_facility = 'LOCAL0'
#syslog_ident = 'postgres'
```

Ce qui me permet à chaque redémarrage de ma machine ou du service - donc du serveur - d'avoir un fichier nommé en postgresql-(date)_(heure).log. Par exemple, pour le fichier postgresql-2006-03-14_152348 stocké dans le répertoire /mnt/pgdata/pg_log, je sais que le serveur a redémarré lors de la journée du 14 Mars 2006 à 15 :23 :48.

1.2.11 Optionnel : Pouvoir créer tous les paquets .deb nécessaires sous Debian

Il y aura surement une manipulation à faire dans le fichier qui répertorie les mirroirs pour les deb de debian /etc/apt/sources.list

Rendez vous ensuite dans le répertoire de root

```
cd /root/
```

Faites ensuite

```
#
# Mise à jour
#
apt-get update
#
# Recherche des .deb sur la 8.1
#
apt-cache search postgresql | grep 8.1
[....]
postgresql-8.1 - object-relational SQL database, version 8.1 server
[....]
```

```
postgresql-server-dev-8.1 - development files for PostgreSQL 8.1 server-side programming
#
# Compilation pour la 8.1 à partir des sources qui seront téléchargées et les .deb ↩
    construites
#
apt-get -b source postgresql-8.1
#
# Liste des différentes deb construites
#
ls
#
# On ontiendra des paquets parmi les suivants
#libecpg5_8.1.0-3_i386.deb          postgresql-client-8.1_8.1.0-3_i386.deb
#libecpg-compat2_8.1.0-3_i386.deb   postgresql-contrib-8.1_8.1.0-3_i386.deb
#libecpg-dev_8.1.0-3_i386.deb       postgresql-doc-8.1_8.1.0-3_all.deb
#libpgtypes2_8.1.0-3_i386.deb       postgresql-plperl-8.1_8.1.0-3_i386.deb
#libpq4_8.1.0-3_i386.deb            postgresql-plpython-8.1_8.1.0-3_i386.deb
#libpq-dev_8.1.0-3_i386.deb         postgresql-pltcl-8.1_8.1.0-3_i386.deb
#postgresql-8.1_8.1.0-3_i386.deb    postgresql-server-dev-8.1_8.1.0-3_i386.deb
#
# Voir le contenu de la .deb principale
#
dpkg -c postgresql-server-dev-8.1_8.1.0-3_i386.deb | less
#
# Installation
#
dpkg -i postgresql-server-dev-8.1_8.1.0-3_i386.deb
```

1.3 Geos et Proj

Nous allons maintenant nous intéresser à l'installation de Geos et de Proj

1.3.1 Téléchargement

Téléchargez les sources respectives de Geos et de Proj grâce à leurs URL respectives

http ://geos.refractions.net/geos-2.2.1.tar.bz2

ftp ://ftp.remotesensing.org/proj/proj-4.4.9.tar.gz

et copiez-les respectivement vers **`/mnt/sources/Geos`** et/**`mnt/sources/Proj`**

1.3.2 Compilations et Installations

Depuis votre terminal, tapez les commandes suivantes. Pour la compilation de Geos, un café ne serait pas de trop ! ! !

```
cd /mnt/sources/Geos

tar xvjf geos-2.2.1.tar.bz2

cd geos-2.2.1

./configure && make && make install

cd /mnt/sources/Proj

tar xvzf proj-4.4.9.tar.gz
```

```
cd proj-4.4.9

./configure && make && make install
```

Normalement le répertoire `/usr/local/bin` doit déjà être accessible par votre variable `PATH`. Voyez celà en l'affichant

```
 env | grep PATH=
```

qui devrait renvoyer quelque chose comme

```
PATH=...;/usr/local/bin
```

Si ce n'est pas le cas, veuillez alors ouvrir - comme pour PostgreSQL - le fichier `/etc/profile` et écrire

```
PATH=.../usr/local/bin:$PATH
```

Mettons à jour notre variable en faisant

```
source /etc/profile
```

Testons maintenant si Geos et Proj ont été installé convenablement on testant respectivement les commandes suivantes

```
geos-config --version
```

renvoyant

```
2.2.1
```

pour Proj, la commande suivante

```
proj
```

devant renvoyer

```
Rel. 4.4.9, 29 Oct 2004
usage: proj [ -beEfiIlormsStTvVwW [args] ] [ +opts[=arg] ] [ files ]
```

Il faut aussi que les libraires de Geos et de Proj soient chargés. Pour celà

```
echo /usr/local/lib >> /etc/ld.so.conf
```

puis

```
ldconfig
```

1.4 PostGIS

Nous allons maintenant nous intéresser à l'installation de PostGIS

1.4.1 Téléchargement

Téléchargez les sources de PostGIS version 1.1.2 à cette URL

http ://postgis.refractions.net/download/postgis-1.1.2.tar.gz

et copiez ce fichier vers **`/mnt/sources/PostGIS`**

1.4.2 Compilation et Installation

Depuis votre terminal, commençons par décompresser les sources :

```
cd /mnt/sources/PostGIS

tar xvzf postgis-1.1.2.tar.gz
```

Puis depuis votre terminal, faites

```
cd /mnt/sources/PostGIS/postgis-1.1.2

./configure
```

Il ne reste plus qu'à compiler et installer PostGIS :

```
cd ~/sources/PostGIS/postgis-1.1.2

make && make install
```

Nous verrons l'utilisation de PostGIS en temps utile.

1.4.3 Tests de régression

A partir de la version 1.1.2 de PostGIS, les tests de régression ont fortement été améliorés et mieux maintenues. Les tests de régression lorsqu'une version vient tout juste de sortir ou une prochaine version est sur le point de sortir permettent de vérifier que PostGIS fournira le minima vital attendu pour fonctionner avant toute utilisation ultérieure. Celà permet également de tester les futures nouveautés de la version car qui dit nouvelle version dit aussi nouveaux tests éventuels. En date de la version 1.1.2, 12 tests de régression sont fournis.

NOTE
Concernant les tests de régression, il est également possible de les effectuer avant d'installer PostGIS ("make install") mais la compilation ("make") est au moins nécessaire.

Rendez-vous dans le répertoire des sources de PostGIS. Et exécutez les commandes suivantes

```
cd /mnt/sources/PostGIS/postgis-POSTGIS_VERSION@
export PGUSER=postgres
make -C regress
```

NOTE
La commande `export PGUSER=postgres` nous évitera de déclarer l'utilisateur root en tant que super-utilisateur de PostgreSQL - ce qui pourrait conduire à la mise en danger du serveur et ce qui est vivement déconseillé.

De la dernière commande ci-dessus résultera l'affichage suivant

```
make: entrant dans le répertoire « /mnt/src/PostGIS/postgis-1.1.2/regress »
Creating spatial db postgis_reg

 PostgreSQL 8.1.1 on i686-pc-linux-gnu, compiled by GCC gcc (GCC) 4.0.3 20051201 ( ↩
    prerelease) (Debian 4.0.2-5)
 Postgis 1.1.2 - 2006-04-21 00:13:42
   GEOS: 2.2.1-CAPI-1.0.1
   PROJ: Rel. 4.4.9, 29 Oct 2004
```

```
Running tests

           regress: Ok.
     regress_index: Ok.
    lwgeom_regress: Ok.
       regress_lrs: Ok.
       removepoint: Ok.
          setpoint: Ok.
          simplify: Ok.
        snaptogrid: Ok.
            affine: Ok.
       regress_ogc: Ok.
    regress_bdpoly: Ok.
      regress_proj: Ok.

Run tests: 12
Successful: 12
Failed: 0
make: quittant le répertoire « /mnt/src/PostGIS/postgis-1.1.2/regress »
```

Tout est donc OK ! Voilà vous disposez d’un serveur complet et digne de ce nom.

Chapitre 2

PostGIS

Mon but ici n'est pas de m'étendre sur l'utilisation de PostGIS mais juste montrer comment créer une base avec PostGIS et un éventail de requêtes possibles avec PostGIS et un cas pratique d'utilisation avec MapServer. Nous commencerons donc par voir comment créer une base ayant les fonctionnalités de PostGIS.

Deux séries de requêtes seront ensuite abordées. La première consiste en un survol rapide des possibiltés de PostGIS. Elle a plutôt un aspect ludique -LOL - La deuxième tente de voir plusieurs aspects possibles : importation de données, comportement de PostGIS, requêtes jugées pertinentes. Pour ces deux séries, les index spatiaux Gist ne seront pas abordés -car ici le nombre de données n'est pas assez volumineux-. Des clins d'oeil à MapServer sont fournis dans la deuxième série parfois. Ces derniers sont mis ici en illustration afin de montrer la requête adéquate pour faire la liaison entre PostGIS et MapServer.

L'utilisation avec MapServer - qui clôture ce chapitre - est un cas pratique où l'on doit dans un premier temps importer des données SIG réelles et extraire de ces données une nouvelle table qui puisse être exploitable par MapServer. Dans un second temps, il est demandé de créer de nouvelles données à partir des premières données, de les afficher ensuite dans MapServer. Celà est fait dans l'optique de montrer le travail qui peut attendre un administrateur PostgreSQL/PostGIS.

NOTE
L'installation de MapServer sur Debian ou GNU/Linux n'est pas abordée ici. Merci de vous référer à la documentation que vous trouverez sur le site de postgis.fr pour une éventuelle installation.

2.1 Créer une base avec PostGIS

La création d'une base - par exemple nommée madatabase - se fait en tapant depuis un terminal tout en étant logué sous postgres :

```
createdb madatabase
createlang plpgsql madatabase
psql -d madatabase -f /usr/local/pgsql/share/lwpostgis.sql
psql -d madatabase -f /usr/local/pgsql/share/spatial_ref_sys.sql
```

Examinons en détail les diverses commandes utilisées :

1. La première (**createdb** ...) est la commande utilisée pour créer une base madatabase.
2. PostGIS est écrit en C/C++ Les fonctions spatiales sont stockées dans une librairie `liblwgeom.so.1.1`. PostgreSQL accède à ces fonctions à condition de lui spécifier le langage PL/PGSQL. La seconde commande permet donc de doter notre base de ce langage.
3. Les définitions des diverses fonctions spatiales sont stockées dans le fichier `lwpostgis.sql` que doit accepter notre base. On charge les fonctions spatiales de PostGIS grâce à la troisième commande. On assure donc un pont entre notre base de données et la librairie `liblwgeom.so.1.1`

4. Il est souvent utile de pouvoir passer d'un système de projection à un autre et même impératif de se doter des divers systèmes de projections connus (Lambert I Carto, Lambert II Etendu ...).La quatrième commande nous permet de se doter des divers systèmes. Ces derniers sont stockés dans une table par le biais du chargement du fichier `spatial_ref_sys.sql` - nom que portera aussi la table des systèmes de projection -.

NOTE
Comme il existe une variable d'environnement propre à PostgreSQL `PGHOME` - qui ici vaut pour rappel /usr/local/pgsql -, on accède à sa valeur en la préfixant par $

```
createdb madatabase
createlang plpgsql madatabase
export PGHOME=/usr/local/pgsql
psql -d madatabase -f $PGHOME/share/lwpostgis.sql
psql -d madatabase -f $PGHOME/share/spatial_ref_sys.sql
```

2.2 Effectuer des requêtes : le moniteur interactif psql de PostgreSQL

`psql` - contenu dans toute distibrution même celle que nous avons compilée et installée - est le moniteur interactif de PostgreSQL. Il permet en outre de faire des requêtes adresser directement à un serveur PostgreSQL. On se connecte à une base `mabasededonnees` en faisant

```
psql mabasededonnees
```

NOTE
Pour sortir du moniteur interactif, il suffit de saisir **\q**

2.3 Effectuer des requêtes : PgAdmin III

PgAdmin est de loin le meilleur outil libre qui soit pour effectuer ses requêtes sous PostgreSQL. Mais il peut faire bien mieux. Le site de pgadmin est http ://www.pgadmin.org

NOTE
Pour ma part c'est avec psql qu'aura lieu les requêtes de ce chapitre.Sur le site de PgAdmin, un paquet .deb existe déjà pour debian - merci de suivre les informations détaillées sur le site, si vous souhaitez l'installer.

2.4 Exemples de requêtes spatiales I

2.4.1 Création de la base et d'une table

Ici nous allons commencer par créer une base de données testgis que nous allons ensuite peupler par quelques données basiques. Pour la création de la base, reportez-vous à la première section de ce chapitre, ce qui depuis un terminal devrait nous donner comme commandes à exécuter

```
createdb testgis
createlang plpgsql testgis
psql -d testgis -f /usr/local/pgsql/share/lwpostgis.sql
psql -d testgis -f /usr/local/pgsql/share/spatial_ref_sys.sql
```

Connectons-nous maitenant à notre base testgis grâce à la commande suivante :

```
psql testgis
```

Peuplons maintenant notre base par quelques données génériques. Pour cela, créons une petite table `test` en saisissant

```
CREATE TABLE test (id serial PRIMARY KEY,genre text);
```

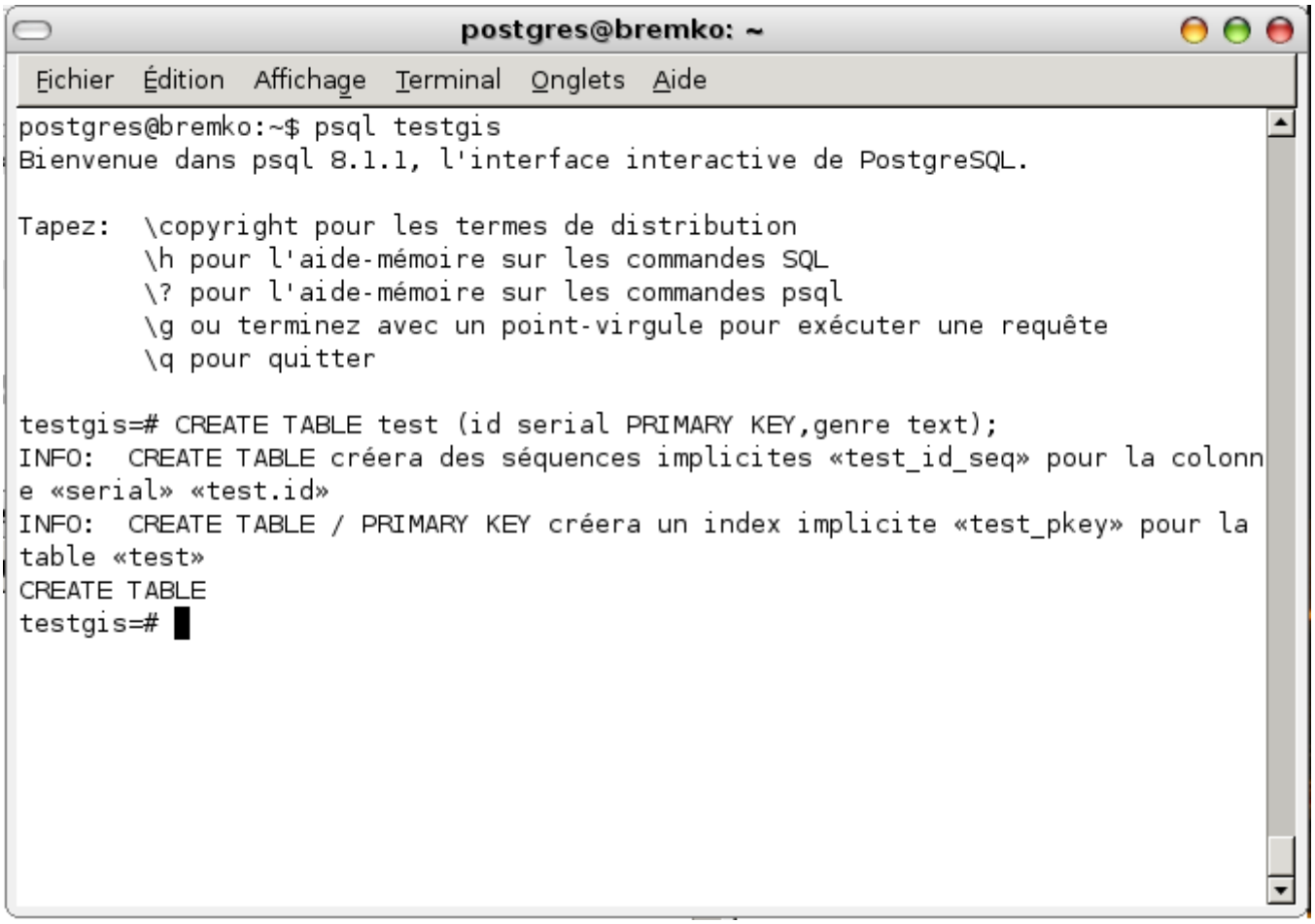

FIG. 2.1 – Le moniteur psql - Connexion à la base testgis.

2.4.2 Ajout de la colonne géométrique à la table - AddGeometryColumn()

Pour l'instant notre table est vide, nous allons lui ajouter une troisième colonne où seront stockées nos données géométriques. Nous aurons recours pour cela à la fonction `AddGeometryColumn()` de PostGIS dont la synthaxe générale est

```
AddGeometryColumn( [Table],
                   [Colonne_Geometrique],
                    [SRID],
                    [Type_Geometrie],
                    [Dimension]);
```

où

[Table] est le nom de la table à laquelle doit être ajoutée la colonne géométrique ;

[Colonne_Geometrique] est le nom de la colonne géométrique ;

[Type_Geometrie] est le le type de géométrie possible :

[SRID] est l'identifiant spatial de projection selon le système de projection choisi. A titre d'exemple pour le Lambert II Carto Etendu, srid=27852 ;

[Dimension] est la dimension des objets géométriques 2D ou 3D ou 4D ;

Dans notre cas, nous saisirons

```
SELECT AddGeometryColumn( 'test', 'geom', -1, 'GEOMETRY', 2 );
```

Nous avons avec cette requête ajouté la colonne géométrique geom ([Colone_Geometrique]) à la table test ([Table]). Comme nous utilisons ici aucun système de projection référencé, - nous restons donc dans le plan 2D orthonormal par défaut - nous précisons juste srid=-1([SRID]). Le type de données géométriques que nous souhaitons enregistrer peut-être de n'importe quel type. Nous préciserons donc par défaut GEOMETRY ([Type_Geometrie]). Nous sommes en 2D ([Dimension]).

NOTE

Il aurait été tout à fait possible de créer la table test en faisant directement

```
CREATE TABLE test (id serial PRIMARY KEY,genre TEXT,geom GEOMETRY);
```

Mais il y aurait alors eu un perte concernant les méta-données sur les objets géométriques concernant le type, le srid, la dimension. Il est donc préférable d'avoir recours à la fonction AddGeometryColumn() pour pouvoir tirer profit de ces métadonnées.

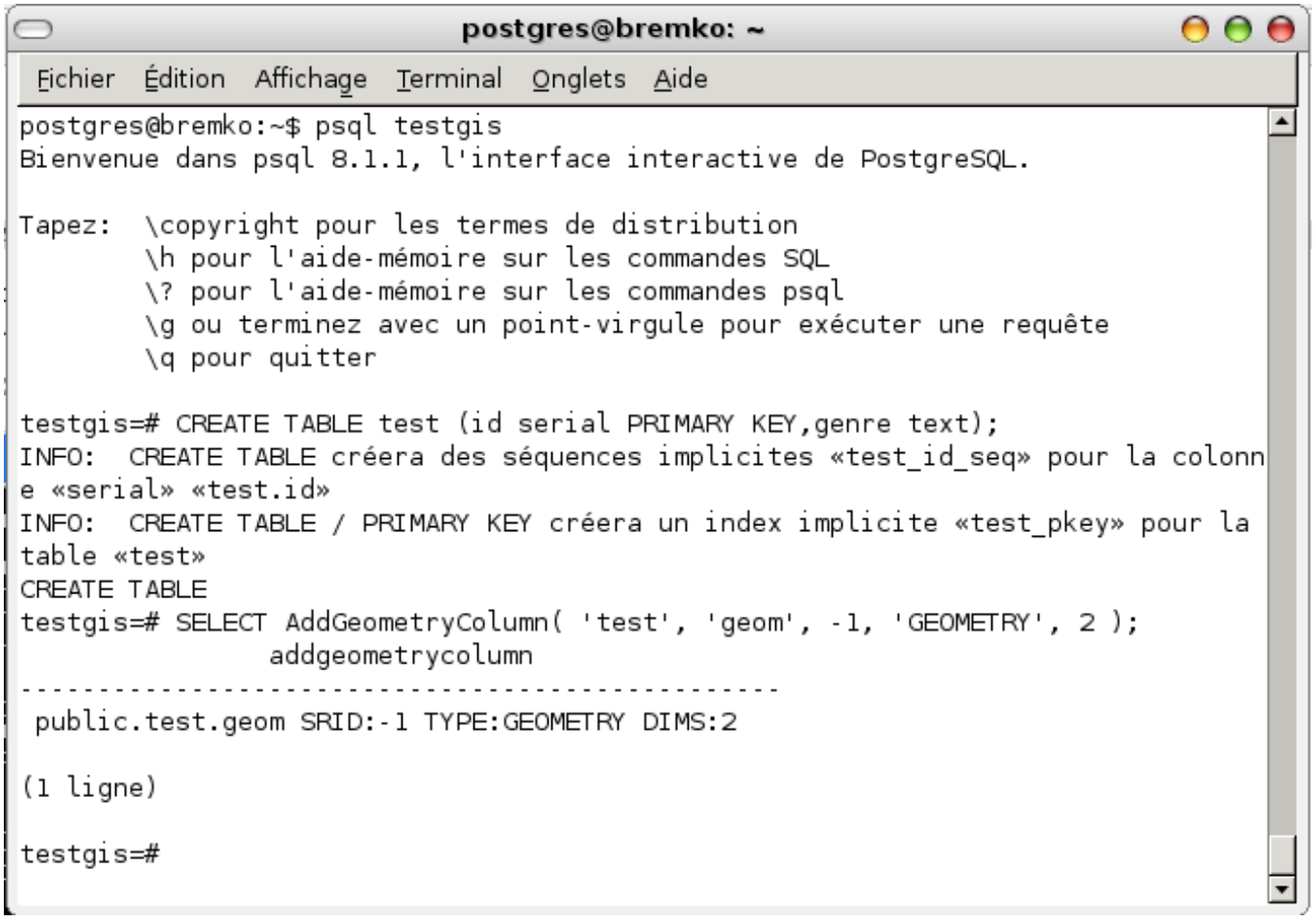

FIG. 2.2 – psql : Fonction AddGeometryColumn() de PostGIS

2.4.3 Objets géométriques spécifiés par l'O.G.C dans PostGIS

PostGIS supporte amplement les objets géométriques définis par l'O.G.C (Open GIS COnsortium). A titre d'exemple, nous citerons donc :

POINT ;
LINESTRING ;
POLYGON ;
MULTIPOINT ;
MULTILINETRING ;
MULTIPOLYGON ;
GEOMETRYCOLLECTION

Insérons maintenant quelques objets dans notre table

2.4.4 Insertion d objets géometriques - GeometryFromText()

L'insertion d'objets géométriques peut par exemple avoir lieu en ayant recours à la fonction `GeometryFromText()` dont la synthaxe générale est

```
GeometryFromText( [Objet_Geometrique],
                  [SRID])
```

où

[Objet_Geometrique] est l'objet à insérer est qui est "humainement lisible", j'entends par humainement lisible au sens qu'il est au format WKT (Well-Known Text) ;

[SRID] l'identifiant de projection qui doit être conforme à celui choisi la colonne géométrique (ici colone geom avec srid=-1).

NOTE
Il existe aussi une autre définition de la fonction GeometryFromText(). Il est aussi possible d'utiliser les fonctions `GeomFromText()` ou `GeomFromEWKT()` etc...Par exemple pour insérer l'objet 'POINT(10 70)' ayant comme identifiant de projection SRID=-1, les 4 fonctions suivantes sont équivalentes

```
SELECT GeometryFromText( 'POINT(10 70)', -1 );
SELECT GeometryFromText('SRID=-1;POINT(10 70)');
SELECT GeomFromText('SRID=-1;POINT(10 70)');
SELECT GeomFromEWKT('SRID=-1;POINT(10 70)');
```

2.4.5 Insertion des données dans la table

Insérons maintenant quelques objets géométriques. Ici les commandes sont à saisir au fur et à mesure pour chaque commande de type INSERT INTO... ;

```
INSERT INTO test VALUES ( 1,
'pieton 1', GeometryFromText( 'POINT(10 70)', -1 ) );

INSERT INTO test VALUES ( 2,
'pieton 2', GeometryFromText( 'POINT(30 30)', -1 ) );

INSERT INTO test VALUES ( 3,
'batiment 1', GeometryFromText( 'POLYGON((10 10,40 20,35 8,12 4,10 10))', -1 ) );

INSERT INTO test VALUES ( 4,
'batiment 2',
GeometryFromText( 'POLYGON((10 40,20 30,30 40,40 35,50 60,35 80,20 60,10 40))', -1 ) );
```

```
INSERT INTO test VALUES ( 5,
'batiment 3', GeometryFromText( 'POLYGON((10 95,20 95,20 135,10 135,10 95))', -1 ) );

INSERT INTO test VALUES ( 6,
'pieton 3', GeometryFromText( 'POINT(35 70)', -1 ) );

INSERT INTO test VALUES ( 7,
'pieton 4', GeometryFromText( 'POINT(35 60)', -1 ) );

INSERT INTO test VALUES ( 8,
'bordure 1 route', GeometryFromText( 'LINESTRING(1 85,50 85)', -1 ) );

INSERT INTO test VALUES ( 9, 'bordure 2 route', GeometryFromText( 'LINESTRING(1 92,50 92)', ↩
    -1 ) );
```

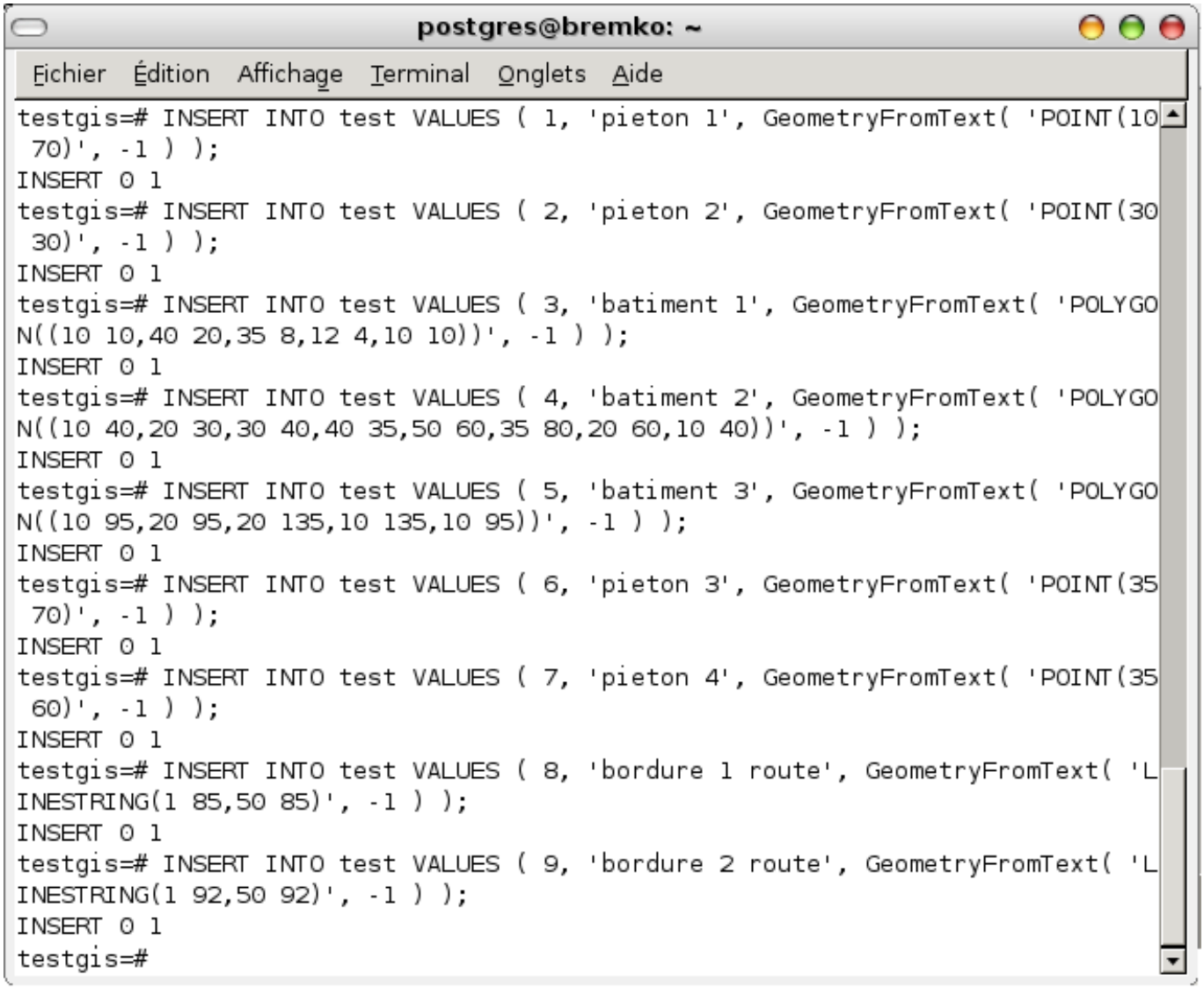

FIG. 2.3 – psql : Insertion des données dans la table test

Le visuel de la table test est le suivant

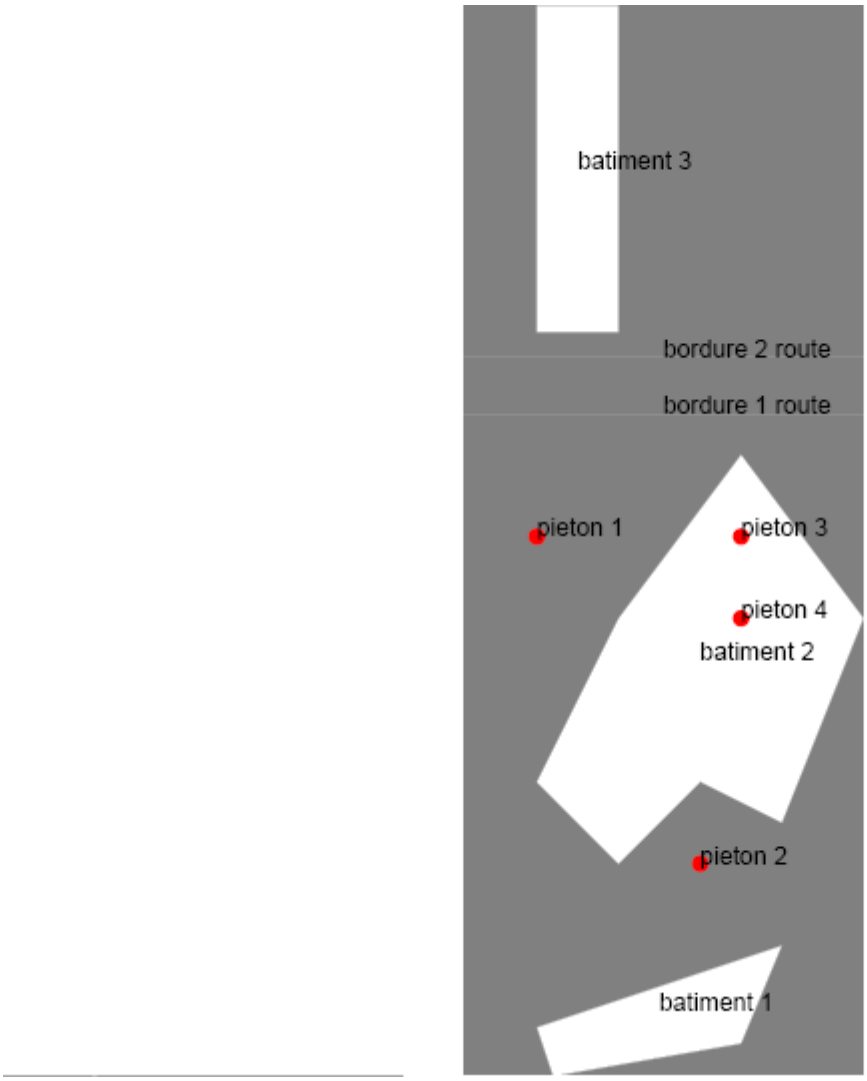

FIG. 2.4 – Visualisation de la table test dans la base testgis

Les instructions/requêtes suivantes doivent être saisie dans le terminal de psql.

NOTE
Il est tout à fait possible de saisir les requêtes sans entrer dans le terminal de psql.Pour cela, il suffit de saisir

```
psql -d testgis -c "VOTRE_REQUETE"
```

2.4.6 Question : Quelles sont les aires des objets ? - Area2d() -

La fonction `Area2d()` renvoie les aires des objets et la

```
testgis=# SELECT id,genre,Area2d(geom) FROM test;
 id |      genre       | area2d
----+------------------+--------
  1 | pieton 1         |      0
  2 | pieton 2         |      0
  3 | batiment 1       |    228
```

```
  4 | batiment 2      |   1050
  5 | batiment 3      |    400
  6 | pieton 3        |      0
  7 | pieton 4        |      0
  8 | bordure 1 route |      0
  9 | bordure 2 route |      0
(9 rows)
```

2.4.7 Question : Quel sont les types géométriques des objets ? - GeometryType() -

Il faut pour cela utiliser la fonction `GeometryType()`

```
testgis=# SELECT id,genre,geometrytype(geom) FROM test;
 id |      genre      | geometrytype
----+-----------------+--------------
  1 | pieton 1        | POINT
  2 | pieton 2        | POINT
  3 | batiment 1      | POLYGON
  4 | batiment 2      | POLYGON
  5 | batiment 3      | POLYGON
  6 | pieton 3        | POINT
  7 | pieton 4        | POINT
  8 | bordure 1 route | LINESTRING
  9 | bordure 2 route | LINESTRING
(9 rows)
```

2.4.8 Question : Qui est dans le bâtiment 2 ? - Distance() -

Ici, il suffira par exemple de trouver les objets dont le champs genre commence par pieton... et de déterminer les objets dont la distance à ce bâtiment est nulle :

```
testgis=# SELECT genre AS pietons_dans_batiment_2 FROM test
WHERE
       Distance((SELECT geom FROM test WHERE genre LIKE 'batiment 2'),test.geom)=0
AND
      genre LIKE 'pieton%';
 pietons_dans_batiment_2
-------------------------
 pieton 3
 pieton 4
(2 rows)
```

2.4.9 Question : Qui est dans le bâtiment 2 ? - WithIn() -

Ici, je propose par rapport à la première façon de faire d'utiliser la fonction `WithIn(A,B)` qui permet de savoir si A est contenu dans B

```
testgis=# SELECT genre AS pietons_dans_batiment_2 FROM test
WHERE
      WithIn(test.geom,(SELECT geom FROM test WHERE genre LIKE 'batiment 2'))
 AND
      genre like 'pieton%';
 pietons_dans_batiment_2
-------------------------
 pieton 3
 pieton 4
(2 rows)
```

2.4.10 Question : Quel est l'objet géométrique le plus proche du pieton 2 ? - Min(), Distance() -

Pour cette question, nous aurons recours à la fonction `Min()` et à la fonction `Distance()`

```
testgis#SELECT q.genre FROM test z,test q WHERE z.genre LIKE 'pieton 2' AND
Distance(z.geom,q.geom)=(SELECT min(foo.valeur) FROM (SELECT h.genre AS nom,distance(t.geom ↩
    ,h.geom)
AS valeur FROM test t, test h WHERE t.genre LIKE 'pieton 2' AND h.genre<>'pieton 2') foo);
   genre
------------
 batiment 2
(1 row)
```

Plusieurs formulations sont possibles pour les requêtes, on aurait ici pu aussi proposer la requête suivante

```
SELECT b.genre,Distance(a.geom,b.geom) FROM test a,test b
  WHERE a.genre='pieton 2'
  AND a.genre!=b.genre
   ORDER BY  distance ASC limit 1;
```

qui donnera le résultat suivant attendu

```
   genre    |     distance
------------+------------------
 batiment 2 | 7.07106781186548
(1 ligne)
```

2.5 Exemples de requêtes spatiales II

Je ne propose ici que des exemples de bases de requêtes spatiales. Pour tirer un meilleur profit possible des fonctions spatiales, je ne serais vous conseiller la documentation sur le site de PostGIS réalisé par Paul RAMSEY. Pour de plus amples informations sur PostgreSQL, n'hésitez pas non plus à regarder dans le répertoire **/usr/local/pgsql/doc/html.**

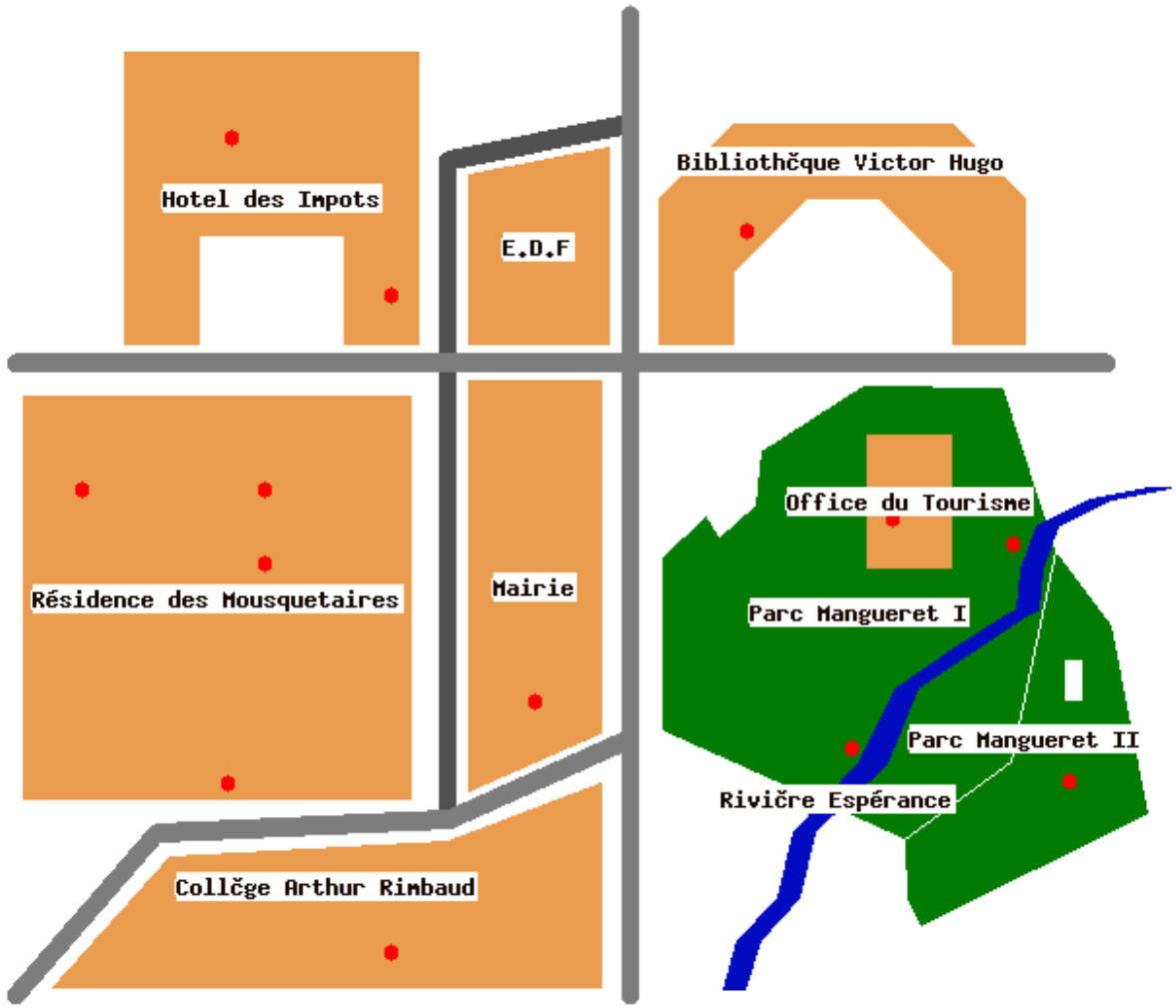

FIG. 2.5 – Visualisation des bâtiments (tables buildings et parcs et rivers).

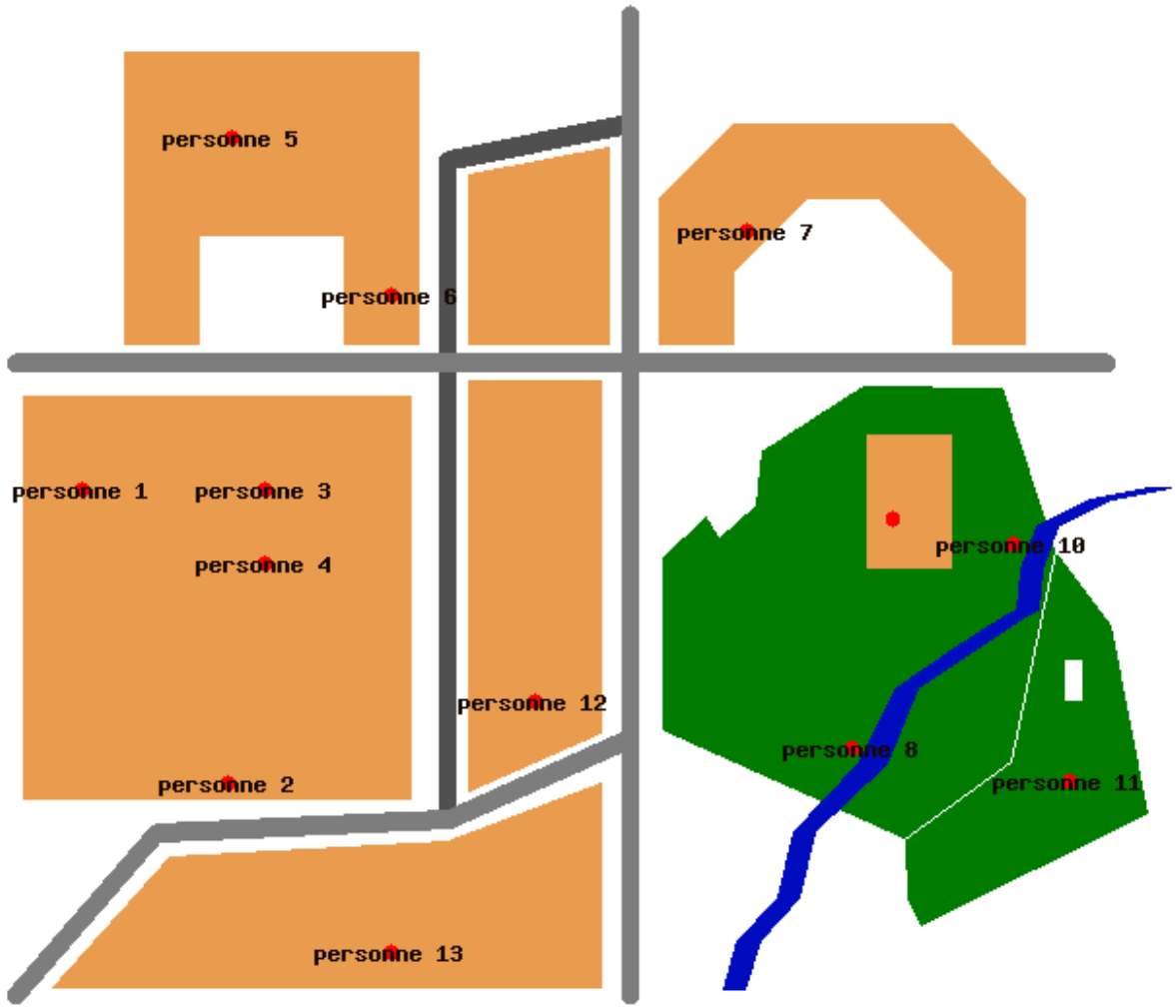

FIG. 2.6 – Visualisation des personnes (table personnes).

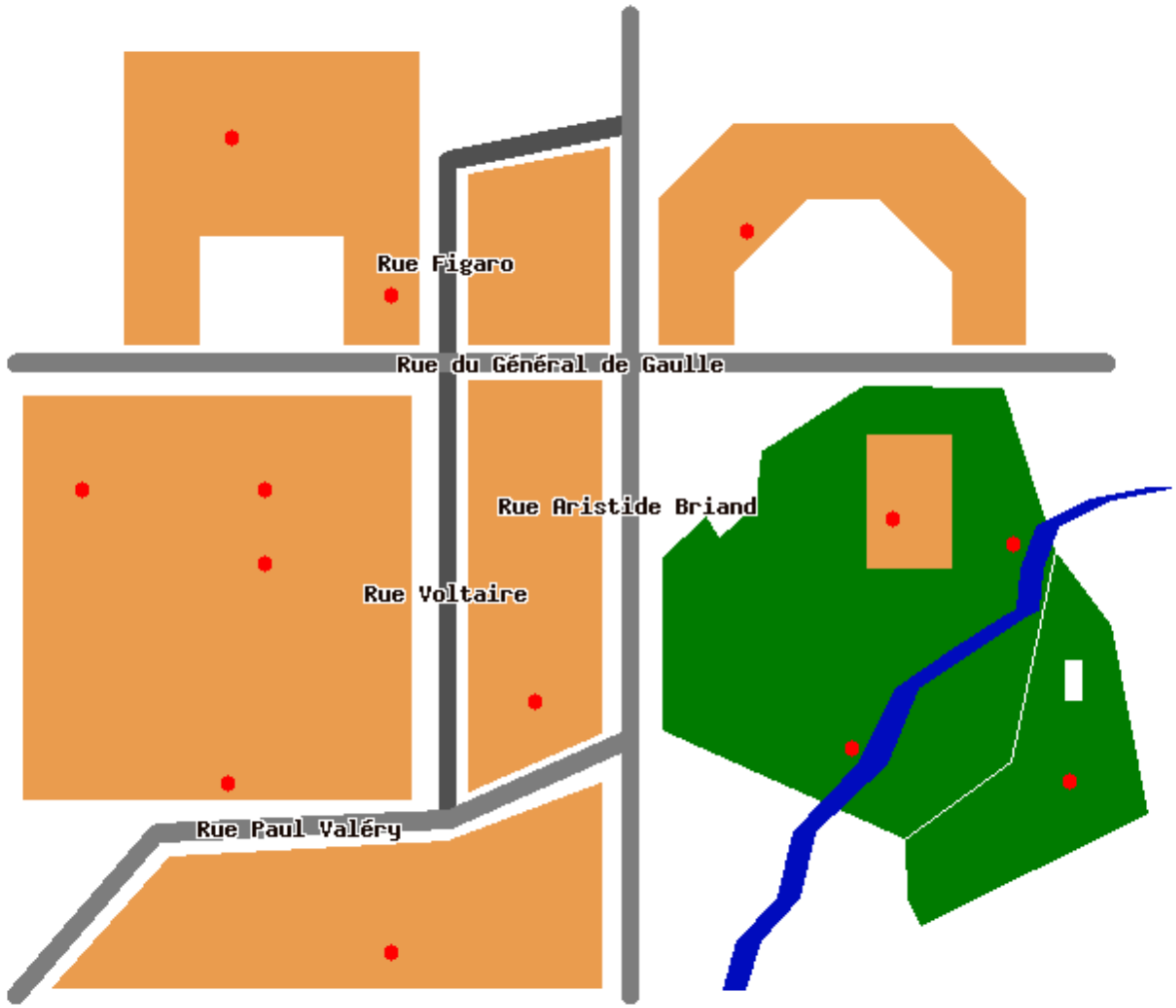

FIG. 2.7 – Visualisation des petites et grandes routes (tables small_roads et great_roads).

2.5.1 Chargement des données par SQL

Pour charger le contenu du fichier sql suivant, copiez-le dans un fichier `madatabase.sql` et depuis un terminal, tapez

```
psql -d mdatabase -f madatabase.sql
```

Le contenu du script `madatabase.sql` [1]est

```
/*
================================================================================
      TECHER Jean David

    Script SQL 'madatabase.sql' à utiliser avec la base madatabase

   Toutes les tables ont la même struture à savoir les champs suivants
```

[1]Ce script est aussi disponible à l'adresse http ://www.postgis.fr/download/win32/madatabase.zip

```
   - id : identifiant pour l'objet

   - data: qui est la donnée attributaire de l'objet

   - the_geom: qui est l'objet spatial

   Il y a en tout 6 tables (voir commentaires pour chaque table).
================================================================================
*/
/*
================================================================================

    -  commentaires:...... .... -

================================================================================
*/
/*
================================================================================

   Cette fonction écrite en SQL permet de tester si une table existe et de
   l'effacer le cas échéant.

================================================================================
*/
CREATE OR REPLACE FUNCTION drop_table_if_exists(text, bool) RETURNS text AS '
DECLARE
opt text;
rec record;
BEGIN
IF $2 THEN
opt := '' CASCADE'';
ELSE
opt := '''';
END IF;

IF NULLVALUE($1) THEN

    RETURN ''ATTENTION: Table  non trouvée'';

ELSE

  SELECT INTO rec tablename FROM pg_tables WHERE tablename like $1;

  IF FOUND THEN
    EXECUTE ''DROP TABLE '' || $1 || opt;
    RETURN ''Effacement de la table ''|| $1 || ''...OK'';
  END IF;

END IF;

     RETURN ''ATTENTION: Table ''|| $1 || '' non trouvée'';

END;
' LANGUAGE 'plpgsql';
/*
================================================================================

   On utilise la fonction pour effacer les tables éventuelles.
   Fonction utile au cas où on devrait recharger ce fichier.
   On évitera aussi d'effacer les tables geometry_columns et spatial_ref_sys

================================================================================
```

```
*/
SELECT tablename,drop_table_if_exists(tablename,false)
FROM pg_tables
  WHERE (tablename NOT LIKE 'pg_%')
   AND (tablename NOT LIKE 'spatial_ref_sys' )
   AND (tablename NOT LIKE 'sql%' )
   AND (tablename NOT LIKE 'geom%' );
/*

================================================================================

    Table pietions

    Certains des piétions sont dans certains des bâtiments.

================================================================================
*/
CREATE TABLE personnes (
   id serial PRIMARY KEY,
   data text);
SELECT AddGeometryColumn('','personnes','the_geom','-1','POINT',2);

INSERT INTO personnes VALUES ( 1, 'personne 1', GeometryFromText( 'POINT(10 70)', -1 ) );
INSERT INTO personnes VALUES ( 2, 'personne 2', GeometryFromText( 'POINT(30 30)', -1 ) );
INSERT INTO personnes VALUES ( 3, 'personne 3', GeometryFromText( 'POINT(35 70)', -1 ) );
INSERT INTO personnes VALUES ( 4, 'personne 4', GeometryFromText( 'POINT(35 60)', -1 ) );
INSERT INTO personnes VALUES ( 5, 'personne 5', GeometryFromText( 'POINT(30.54 118.28)', -1 ↩
    ) );
INSERT INTO personnes VALUES ( 6, 'personne 6', GeometryFromText( 'POINT(52.36 96.73)', -1  ↩
   ) );
INSERT INTO personnes VALUES ( 7, 'personne 7', GeometryFromText( 'POINT(100.94 105.44)',  ↩
   -1 ) );
INSERT INTO personnes VALUES ( 8, 'personne 8', GeometryFromText( 'POINT(115.16 34.81)', -1 ↩
    ) );
INSERT INTO personnes VALUES ( 9, 'personne 9', GeometryFromText( 'POINT(120.89 66.23)', -1 ↩
    ) );
INSERT INTO personnes VALUES ( 10, 'personne 10', GeometryFromText( 'POINT(137.40 62.56)',  ↩
   -1 ) );
INSERT INTO personnes VALUES ( 11, 'personne 11', GeometryFromText( 'POINT(144.97 30.23)',  ↩
   -1 ) );
INSERT INTO personnes VALUES ( 12, 'personne 12', GeometryFromText( 'POINT(72.04 41.23)',  ↩
   -1 ) );
INSERT INTO personnes VALUES ( 13, 'personne 13', GeometryFromText( 'POINT(52.32 6.84)', -1 ↩
    ) );
/*
================================================================================

  Table buildings:

   J'ai opté pour des noms classiques de bâtiments administratifs

================================================================================
*/
CREATE TABLE buildings (
   id serial PRIMARY KEY,
   data text);
SELECT AddGeometryColumn('','buildings','the_geom','-1','POLYGON',2);

INSERT INTO buildings VALUES ( 1, 'Collège Arthur Rimbaud',
GeometryFromText( 'POLYGON((6 2,81 2,81 30,60 22,22 20,6 2))', -1 ) );

INSERT INTO buildings VALUES ( 2, 'Résidence des Mousquetaires',
```

```
GeometryFromText( 'POLYGON((2 83,55 83,55 28,2 28,2 83))', -1 ) );

INSERT INTO buildings VALUES ( 3, 'Hotel des Impots',
GeometryFromText( 'POLYGON((16 90,26 90,26 105,46 105,
46 90,56 90,56 130,16 130,16 90))', -1 ) );

INSERT INTO buildings VALUES ( 4,'E.D.F',
GeometryFromText( 'POLYGON((63 90,82 90,82 117,63 113.15,63 90))', -1 ) );

INSERT INTO buildings VALUES ( 5,'Bibliothèque Victor Hugo',
GeometryFromText( 'POLYGON((89 90,99 90,99 100,109 110,119 110,129 100,
129 90,139 90,139 110,129 120,99 120,89 110,89 90))', -1 ) );

INSERT INTO buildings VALUES ( 6, 'Mairie',
GeometryFromText( 'POLYGON((63 85,81 85,81 37,63 29,63 85))', -1 ) );

INSERT INTO buildings VALUES ( 7, 'Office du Tourisme',
GeometryFromText( 'POLYGON((117.36 77.6,128.71 77.60,128.71 59.49,
117.36 59.49,117.36 77.6))', -1 ) );
/*
================================================================================

  Tables small_roads:

  Il n'y en a que deux.

================================================================================
*/
CREATE TABLE small_roads (
  id serial PRIMARY KEY,
   data text);
SELECT AddGeometryColumn('','small_roads','the_geom','-1','LINESTRING',2);

INSERT INTO small_roads VALUES (1, 'Rue Figaro',
GeometryFromText( 'LINESTRING(60 87.5,60 115,85 120)', -1 ) );

INSERT INTO small_roads VALUES (2, 'Rue Voltaire',
GeometryFromText( 'LINESTRING(60 87.5,60 25)', -1 ) );
/*
================================================================================

   Tables great_roads

   Il y en a 3. Les grandes routes gardent leur nom malgré un ou plusieurs
   croisement avec d'autres routes

================================================================================
*/
CREATE TABLE great_roads (
  id serial PRIMARY KEY,
   data text);
SELECT AddGeometryColumn('','great_roads','the_geom','-1','LINESTRING',2);

INSERT INTO great_roads VALUES (1, 'Rue Paul Valéry',
GeometryFromText( 'LINESTRING(1 1,20 23,60 25,85 36)', -1 ) );
INSERT INTO great_roads VALUES ( 2, 'Rue du Général de Gaulle',
GeometryFromText( 'LINESTRING(1 87.5,150 87.5)', -1 ) );
INSERT INTO great_roads VALUES ( 3, 'Rue Aristide Briand',
GeometryFromText( 'LINESTRING(85 1,85 135)', -1 ) );
/*
================================================================================
```

```
   Table parc

   Il y en a deux portant le nom générique de Mangueret. Le premier parc est
   traversé par la rivière et conteint le bâtiment de l'Office du Tourisme.
   Le deuxième parc contient une aire adminitrative en réaménagement rachétée
   par la mairie. Le temps de la construction cette aire ne fait donc par partie
   du parc. Soit un trou dans le polygone modélisant ce parc.

================================================================================
*/
CREATE TABLE parcs (
  id serial PRIMARY KEY,
   data text);
SELECT AddGeometryColumn('','parcs','the_geom','-1','POLYGON',2);

INSERT INTO parcs VALUES (1,'Parc Mangueret I',
GeometryFromText( 'POLYGON((89.19 37.11,89.19 60.73,
95.31 66.56,97.23 63.68,102.03 68,102.75 75.44,116.91 84.5,136 84.08,
143.08 62,140.92 50,137.08 32.95,122.43 22.39,89.19 37.11))',-1));
INSERT INTO parcs VALUES (2,'Parc Mangueret II',
difference(
GeometryFromText( 'POLYGON((143.08 62,150.94 51.41,155.96 25.66,
124.64 10.38,122.78 14.09,122.43 22.39,137.08 32.95,143.08 62))',-1),
GeometryFromText( 'POLYGON((144.55 46.64,146.68 46.64,146.68 41.46,
144.55 41.46,144.55 46.64))',-1)));
/*
================================================================================

   Table rivers

================================================================================
*/

CREATE TABLE rivers (
  id serial PRIMARY KEY,
   data text);
SELECT AddGeometryColumn('','rivers','the_geom','-1','POLYGON',2);

INSERT INTO rivers VALUES (1,'Rivière Espérance',
GeometryFromText( 'POLYGON((97.71 1.98,99.63 8.46,105.15 14.23,107.31 23.35,
115.95 32.71,121.23 43.03,129.15 48.31,
135.64 52.64,137.80 53.60,138.28 59.36,
140.44 65.12,147.88 68.72,155.80 70.40,158.80 70.40,
150.88 68.72,143.44 65.12,141.28 59.36,140.80 53.60,138.64 52.64,
132.15 48.31,124.23 43.03,119.95 32.71,110.31 23.35,108.15 14.23,
102.63 8.46,100.71 1.98,97.71 1.98))',-1));

/*
================================================================================

    -  Fonction permettant de créer des index spatiaux -

  On va commencer par vérifier quel est le nom de la colonne géométrique
  dans la table demandée (par défaut = the_geom) ...Il faut bien sûr que ce nom soit
  préciser dans la table geometry_columns.

  Cette fonction permet surtout d'alléger la commande de création d'indexation spatiale

================================================================================
*/
CREATE OR REPLACE FUNCTION Create_Gis_Index(text) RETURNS text AS '
DECLARE
```

```
rec record;
BEGIN

SELECT INTO rec f_geometry_column FROM geometry_columns WHERE f_table_name like $1;


  IF FOUND THEN
EXECUTE ''CREATE INDEX '' || $1 || ''_index_spatial ON ''|| $1 ||
'' USING gist(''|| rec.f_geometry_column  || '' gist_geometry_ops)'';
    RETURN ''Index Spatial sur '' || $1 || ''...OK'';
  END IF;


  RETURN ''Impossible de créer index spatial sur '' || $1 ;
END;
' LANGUAGE 'plpgsql';
/*
================================================================================

    -  Création des index spatiaux -

================================================================================
*/
SELECT Create_Gis_Index(tablename)
FROM pg_tables
  WHERE (tablename NOT LIKE 'pg_%')
   AND (tablename NOT LIKE 'spatial_ref_sys' )
   AND (tablename NOT LIKE 'sql%' )
   AND (tablename NOT LIKE 'geom%' )
   ORDER BY tablename;
/*
================================================================================

    -  Vérification de la création des index spatiaux -

================================================================================
*/
\di *_index_spatial
/*
================================================================================

    -  Table de permettant de gérer MapServer -

================================================================================
*/
CREATE TABLE mapserver_desc (
          ms_gid serial PRIMARY KEY,
          ms_table text,
          ms_libelle text,
          ms_name text,
          ms_labelitem text,
          ms_color text,
          ms_outlinecolor text,
          ms_symbol text
);


INSERT INTO mapserver_desc values (0,'small_roads', 'Petite routes',
'small_roads','data','80 80 80','80 80 80',NULL);
INSERT INTO mapserver_desc values (1,'great_roads','Grandes routes',
'great_roads','data','125 125 125','125 125 125',NULL);
INSERT INTO mapserver_desc values (2,'parcs','Parcs publiques',
```

```
'parcs','data','0 123 0','0 123 0',NULL);
INSERT INTO mapserver_desc values (3,'rivers','Rivière',
'rivers','data','0 12 189','0 12 189',NULL);
INSERT INTO mapserver_desc values (4,'buildings', 'Bâtiments',
'buildings','data','234 156 78','234 156 78',NULL);
INSERT INTO mapserver_desc values (5,'personnes', 'personnes',
'personnes','data',NULL,'0 0 255','circle');
```

2.5.2 Chargement de données par ESRI Shapefiles (shp2pgsql)

Nous allons supposer dans cette sous-sections que nous disposons des mêmes sources de données que celles ci-dessus. Ces dernières cette fois-ci au lieux d'être stockées au format sql sont ici stockées au format .shp. Chaque table est ainsi stockées dans un fichier .shp - avec les fichiers auxiliaires adéquates (.dbf, .shx....).

NOTE
Les fichiers en question sont disponibles à http ://www.postgis.fr/download/win32/shp.zip. Décompressez ce fichier(unzip shp.zip) et double-cliquez sur le fichier batch import.bat pour les charger dans la base. N'hésitez pas à ouvrir pour l'étudier par rapport aux explications qui suivent sur shp2pgsql.exe. SI vous n'avez pas pour le moment importé le fichier madatabase.sql (cf. "Chargement en SQL") ouvrez alors ce fichier et remplacez l'option "`-dD`" en "`-D`".

PostGIS est livré avec un convertisseur **shp2pgsql** qui permet d'importer directement - à la volée - les données fournies dans un fichier .shp. Pour de meilleurs informations sur ce convertisseur, il suffit de saisir

```
shp2pgsql --help
```

qui vous renverra en sortie

```
shp2pgsql: illegal option -- -
RCSID: $Id: shp2pgsql.c,v 1.104 2005/11/01 09:25:47 strk Exp $
USAGE: shp2pgsql [<options>] <shapefile> [<schema>.]<table>

OPTIONS:
  -s <srid>  Set the SRID field. If not specified it defaults to -1.

  (-d|a|c|p) These are mutually exclusive options:
      -d  Drops the table , then recreates it and populates
          it with current shape file data.
      -a  Appends shape file into current table, must be
          exactly the same table schema.
      -c  Creates a new table and populates it, this is the
          default if you do not specify any options.
      -p  Prepare mode, only creates the table

  -g <geometry_column> Specify the name of the geometry column
     (mostly useful in append mode).

  -D  Use postgresql dump format (defaults to sql insert
      statments.

  -k  Keep postgresql identifiers case.

  -i  Use int4 type for all integer dbf fields.

  -I  Create a GiST index on the geometry column.

  -w  Use wkt format (for postgis-0.x support - drops M - drifts coordinates).

  -N <policy> Specify NULL geometries handling policy (insert,skip,abort)
```

Avec shp2pgsql, la ligne de commande la plus utilisée est

```
shp2pgsql -D -I shapefile nom_table | psql base_de_donnees
```

où l'option

-D permet d'importer les données sous forme de COPY au lieu de INSERT. Ce qui est beaucoup plus rapide au niveau de l'importation des données

-I permet de créer automatiquement un index spatial pour les données géométriques importées

shapefile est le chemin d'accès complet vers le shapefile

nom_table est le nom que vous souhaiteriez donner à votre table

Dans notre cas nous allons supposer que tous nos shapefiles sont dans le répertoire `c :\Mes donnees SIG\shp` et que les noms respectives des tables seront le même que celui des fichiers .shp. Le contenu en fichier .shp du répertoire en question est obtenu par la commande **ls** depuis un terminal

```
$ ls *.shp
buildings.shp  great_roads.shp  parcs.shp  personnes.shp  rivers.shp  small_roads.shp
```

Pour nos importations, il nous suffira de faire depuis un terminal

pour le chargement :

```
for i in `ls *.shp`;\
> do \
> table=`basename $i | cut -d '.' -f 1`;\
> shp2pgsql -dDI $i $table | psql madatabase;\
> done
```

ou en un peu plus subtil :

```
for i in $(find . | grep shp);do shp2pgsql -dDI $i $(basename $i .shp) | psql  ↩
    madatabase;done
```

qui nous renverra

```
Shapefile type: Polygon
Postgis type: MULTIPOLYGON[2]
               dropgeometrycolumn
-------------------------------------------------
 public.buildings.the_geom effectively removed.
(1 row)

DROP TABLE
BEGIN
NOTICE:  CREATE TABLE will create implicit sequence "buildings_gid_seq" for serial column " ↩
    buildings.gid"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "buildings_pkey" for table " ↩
    buildings"
CREATE TABLE
                                  addgeometrycolumn
-------------------------------------------------------------------------------------
 public.buildings.the_geom SRID:-1 TYPE:MULTIPOLYGON DIMS:2
 geometry_column fixed:0
(1 row)

CREATE INDEX
COMMIT
.
.
.
Shapefile type: Arc
Postgis type: MULTILINESTRING[2]
```

```
                dropgeometrycolumn
-------------------------------------------------
 public.small_roads.the_geom effectively removed.
(1 row)

DROP TABLE
BEGIN
NOTICE:  CREATE TABLE will create implicit sequence "small_roads_gid_seq" for serial column ↩
     "small_roads.gid"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "small_roads_pkey" for table ↩
     "small_roads"
CREATE TABLE
                                    addgeometrycolumn
-------------------------------------------------------------------------------------------
 public.small_roads.the_geom SRID:-1 TYPE:MULTILINESTRING DIMS:2
 geometry_column fixed:0
(1 row)

CREATE INDEX
COMMIT
```

Le message CREATE INDEX confirme la création des index spatiaux Gist, 'addgeometrycolumn ' confirme aussi l'insertion des données spatiales.

2.5.3 Question-Pratique : Qu'elle est la version de PostgreSQL ?

Il suffit pour cela d'utiliser la fonction interne de PostgreSQL : `Version()`

```
select version()
```

qui nous renvoit

```
                                          version
-----------------------------------------------------------------------------------------------
 PostgreSQL 8.1.1 on i686-pc-mingw32, compiled by GCC gcc.exe (GCC) 3.4.2 (mingw-special)
(1 ligne)
```

On peut aussi utiliser la requête suivante

```
SHOW server_version
```

qui nous renvoit

```
 server_version
----------------
 8.1.1
(1 ligne)
```

NOTE
Le numéro de version peut-être aussi obtenu en tapant depuis un terminal :

```
pg_config --version
```

2.5.4 Question-Pratique : Où se trouve notre répertoire de bases de données ? - PGDATA -

La requête suivante

```
SHOW data_directory
```

confime ce que nous avons choisi comme valeur pour la variable d'environnement PGDATA[2]

```
             data_directory
----------------------------------------
 /mnt/pgdata
(1 ligne)
```

2.5.5 Question-Pratique : Qui sont les utilisateurs de PostgreSQL ?

Je ne fais pas ici de distinction entre les super-utilisateurs et les utilisateurs courants de PostgreSQL. En un mot, je ne m'occuperais pas ici des droits attibués (création de base, ajout d'utilisateur etc...). Pour simplifier, on dira que la requête est

```
select usename from pg_user order by usename
```

qui nous renvoit

```
 usename
----------
 postgres
 (1 ligne)
```

On obtient de meilleurs informations en faisant directement :

```
psql -d template1 -c "\du"
```

```
                                  Liste des rôles
 Nom du rôle | Superutilisateur | Crée un rôle | Crée une base | Connexions  | Membre de
-------------+------------------+--------------+---------------+-------------+-----------
 postgres    | oui              | oui          | oui           | sans limite |
(3 lignes)
```

2.5.6 Question-Pratique : Quelles sont les infos sur les outils compilés pour PostGIS ?

La requête suivante

```
select postgis_full_version();
```

nous renverra comme réponse

```
                               postgis_full_version
-------------------------------------------------------------------------------
 POSTGIS="1.1.2" GEOS="2.2.1-CAPI-1.0.1" PROJ="Rel. 4.4.9, 29 Oct 2004" USE_STATS
(1 ligne)
```

Ce qui prouve bien que PostGIS 1.1.2, Geos 2.2.1 et Proj 4.4.9 ont bien compilés.

[2] voir pour cela le chapitre 'Paramétrer PostgreSQL'

2.5.7 Question-Pratique : Quel est le listing des bases de PostreSQL ?

cela se fait en faisant la requête suivante

```
SELECT pg_database.datname
            FROM pg_database, pg_user
          WHERE pg_database.datdba = pg_user.usesysid
          AND pg_database.datname not
          LIKE 'template%' order by datname
```

qui ici renvoit

```
  datname
-----------
madatabase
testgis
(2 lignes)
```

On obtient aussi les mêmes informations en faisant **psql -l**.

2.5.8 Question-Pratique : Quelles sont les tables contenues dans la base ?

La requête suivante

```
select tablename as nom_table from pg_tables where (tablename not like 'pg_%') and
(tablename not like 'spatial_ref_sys' ) and (tablename not like 'sql_%' )
and (tablename not like 'geom%' ) order by tablename; ;
```

nous renverra comme réponse

```
    nom_table
----------------
 buildings
 great_roads
 mapserver_desc
 parcs
 personnes
 rivers
 small_roads
(7 rows)
```

Le listing des tables peut facilement être obtenu en tapant simpelement '\dt' dans psql le moniteur interactif de PostgreSQL.

NOTE
Je pars ici du principe que toutes les tables sont contenues dans un même schéma et soient contenus sur un même espace logique de PostgreSQL. Mais si vous êtes sûr que toutes vos tables sont dans le même schéma, vous pouvez par exemple exploiter la requête

```
select tablename as nom_table from pg_tables where schemaname='public'
and tablename not in ('spatial_ref_sys','geometry_columns') order by tablename;
```

2.5.9 Question-Pratique : Utiliser une vue pour simplifier la recherche du listing des tables.

PostgreSQL offre un concept fort appréciable concernant les SGDBR, celui des vues. Les vues permettent parfois de se simplifier la vie avec des requêtes parfois bien longue et que l'on ne pourrait pas toujours pouvoir se souvenir. Pour la requête concernant le listing des tables contenues dans une base de données - vu précédemment - on peut avoir recours aux vues en créant la la vue suivante

```
create view liste_table as select tablename as nom_table from pg_tables where
(tablename not like 'pg_%') and (tablename not like 'spatial_ref_sys' )
and (tablename not like 'sql_%' ) and (tablename not like 'geom%' ) order by tablename;
```

On aura alors le listing attendu en faisant directement

```
select * from liste_table
```

2.5.10 Question : Où sont stockées les informations relatives aux données spatiales (métadonnées) des tables avec PostGIS ? - table geometry_columns -

Ces informations (srid, type, nom de la colonne géométrique) sont stockées dans la table `geometrycolumns`. La requête

```
select * from geometry_columns
```

nous renvoit les informations suivantes

```
 f_table_catalog | f_table_schema | f_table_name | f_geometry_column | coord_dimension |  ↩
     srid |    type
-----------------+----------------+--------------+-------------------+-----------------+------+-----

                 | public         | personnes    | the_geom          |               2 |    ↩
                    -1 | POINT
                 | public         | buildings    | the_geom          |               2 |    ↩
                    -1 | POLYGON
                 | public         | small_roads  | the_geom          |               2 |    ↩
                    -1 | LINESTRING
                 | public         | great_roads  | the_geom          |               2 |    ↩
                    -1 | LINESTRING
                 | public         | parcs        | the_geom          |               2 |    ↩
                    -1 | POLYGON
                 | public         | rivers       | the_geom          |               2 |    ↩
                    -1 | POLYGON
(6 rows)
```

Les informations utiles sont

f_table_name qui fournit le nom de la table ;

f_geometry_column qui donne le nom de la colonne géométrique ;

srid fournit le srid - identifiant de système de projection- ;

type qui fournit le type d'objet contenu dans la table.

AVERTISSEMENT
Certains logiciels SIG - au sens large du terme - comme MapServer, JUMP ou QGIS exploitent cette table pour effectuer leurs requêtes attendues ou lister l'ensemble des tables exploitables. On comprend ainsi mieux son importance. Cette table ainsi que certains champs de cette table sont aussi spécifiés par l'O.G.C (Oprn GIS Consortium).

2.5.11 Question-Pratique : Comment créer une fonction en PLP/PGSQL qui puisse faire le différentiel entre les tables référencées par geometry_columns et toutes les tables contenus dans le schéma public de la base (tables non géospatiales) ?

Comme il a été fait mention dans la sous-section précédente, la table `geometry_columns` contient toutes les métadonnées nécessaires à la gestion des tables géospatiales de notre base. Or il se peut qu'un jour nous ayons besoin d'une fonction qui puisse faire le différentiel entre les tables géospatiales de notre base avec les tables non géospatiales.

Il est alors possible d'utiliser la fonction suivante

```
create or replace function diff_geometry_columns(OUT j text) as $$
begin
select into j tablename as nom_table  from pg_tables where
   tablename not in ((select f_table_name from geometry_columns)
                      union
                     (select 'spatial_ref_sys') union (select 'geometry_columns'))
and pg_tables.schemaname='public';
end;
$$ language plpgsql;
```

Ainsi l'appel de cette fonction par

```
select diff_geometry_columns();
```

j'obtiens

```
 diff_geometry_columns
-----------------------
 mapserver_desc
(1 ligne)
```

qui est bien la seule table non géospatiale - à part spatial_ref_sys - de ma base.

NOTE
Pour de plus amples informations sur PL/PGSQL, merci de consulter la documentation française de PostgreSQL à http ://traduc.postgresqlfr.org/pgsql-8.1.1-fr/plpgsql.html

---

2.5.12 Question : Comment sont stockées les données géométriques avec PostGIS ?

Avant la version 1.0.0 de PostGIS, les données étaient stockées sous forme WKT (Well-Known Text),c'est-à-dire un format lisible par le commun des mortels. PostGIS prenait aussi en compte les formats géométriques (sémantique, grammaire,tolopolgie...) des objets définis par l'OGC Open GIS Consortium. Pour des raison d'économie sur le disque de dur et pour un accès plus rapides, depuis la version 1.0.0, les formates acceptés par PostGIS sont les formats EWKB /EWKT. Depuis cette version, les formats supportés par PostGIS forment une extension des définitions des formats définis par l'OGC qui ne s'intéressaient qu'aux objets définis en 2D. On peut donc avoir des données 3d. Ainsi EWKB/EWKT sont une extension du WKB/WKT. WKB.WKT sont donc acceptés par PostGIS en tant que EWKB/EWKT en dimension 2D.

NOTE
Pour de plus amples informations, veuillez consulter la documentation de PostGIS.

L"accès direct aux objets géométriques en EWKB ne permet pas de lire leur contenus des données pour le commun des mortels. Je rappelle au passage qu'un objet spatial dans une base de données est défini par son type, l'ensemble ou les sous-ensembles de points qui le composent ainsi qu'au système de projection auquel il est attaché, et à la dimension (2d,3d ou 4d) auquel il appartient. PostgreSQL de son côté selon les processus internes de fonctionnement et les format d'entrée des objets spatiaux en mode ascii ou binaire choisi - ascii : HEXEWKB/EWT et binaire : EWKB - stockera respectievement en ascii : HEXEWKB et pour le binaire en EWKB.

Pour pouvoir respectivement lire les données aux formats EWKT - format lisible -, il faut avoir recours aux fonctions AsText() ou AsEWKT(), et Srid() pour connaître leur identiifant de système de projection. Par exemple pour la table personnes dont la colonne géométrique est the_geom, affichons ces diverses informations grâce à la requête suivante

```
SELECT AsText(the_geom), Srid(the_geom),the_geom FROM personnes
```

qui nous renvoit

```
        astext         | srid |                   the_geom
-----------------------+------+--------------------------------------------
 POINT(10 70)          |   -1 | 0101000000000000000000244000000000000805140
 POINT(30 30)          |   -1 | 010100000000000000000003E400000000000003E40
 POINT(35 70)          |   -1 | 0101000000000000000008041400000000000805140
 POINT(35 60)          |   -1 | 0101000000000000000008041400000000000004E40
 POINT(30.54 118.28)   |   -1 | 01010000000AD7A3703D8A3E4052B81E85EB915D40
 POINT(52.36 96.73)    |   -1 | 0101000000AE47E17A142E4A401F85EB51B82E5840
 POINT(100.94 105.44)  |   -1 | 01010000005C8FC2F5283C59405C8FC2F5285C5A40
 POINT(115.16 34.81)   |   -1 | 01010000000AD7A3703DCA5C4048E17A14AE674140
 POINT(120.89 66.23)   |   -1 | 0101000000295C8FC2F5385E401F85EB51B88E5040
 POINT(137.4 62.56)    |   -1 | 0101000000CDCCCCCCCC2C614048E17A14AE474F40
 POINT(144.97 30.23)   |   -1 | 0101000000D7A3703D0A1F62407B14AE47E13A3E40
 POINT(72.04 41.23)    |   -1 | 0101000000C3F5285C8F0252403D0AD7A3709D4440
 POINT(52.32 6.84)     |   -1 | 0101000000295C8FC2F5284A405C8FC2F5285C1B40
(13 rows)
```

On voit bien ici le stockage réalisé en HEXEWKB des objets géométriques. En effet, si nous effectuons la requête suivante par exemple relatif au premier enregistrement pour le point POINT(35 70) ayant SRID=-1, on a comme attendu en utilisant la fonction `GeomFromEWKT()`

```
SELECT GeomFromEWKT('SRID=-1;POINT(35 70)');
```

ou la requête équivalente

```
SELECT 'SRID=-1;POINT(35 70)'::geometry;
```

on obtient

```
                geomfromewkt
--------------------------------------------
 0101000000000000000008041400000000000805140
(1 row)
```

PostgreSQL lit en entrée (mode ascii) du EWKT et restitue en stockage HEXEWKB (= EWKB sous forme hexadécimale)

Cet objet stocké en HEXEWKB a la forme suivante en EWKT, restitué grâce à la fonction `AsEWKT()`

```
select AsEWKT('0101000000000000000008041400000000000805140');
```

```
    asewkt
--------------
 POINT(35 70)
(1 ligne)
```

et en EWKB restitué grâce à `AsEWKB()`

```
 select AsEWKB('0101000000000000000008041400000000000805140');
```

```
                                asewkb
-----------------------------------------------------------------------
 \001\001\000\000\000\000\000\000\000\000\200A@\000\000\000\000\000\200Q@
(1 ligne)
```

et comme srid, restitué grâce à la fonction `Srid()`

```
select Srid('0101000000000000000008041400000000000805140'::geometry);
```

```
 srid
------
   -1
(1 ligne)
```

En mode d'entrée, PostGIS accepte aussi le HEXEWKB directement et PostgreSQL le stockera tout simplement en HEXEWKB. Ainsi dans dans une table test dont la structure serait par exemple

```
CREATE TABLE test (...,
                  the_geom GEOMETRY);
```

les deux insertions suivantes sont équivalentes

```
INSERT INTO test VALUES (...,
                       GeomFromEWKT('SRID=-1;POINT(35 70)')
                  );
```

```
INSERT INTO test VALUES (...,
                    '0101000000000000000008041400000000000805140'
                   );
```

Ici il n'est pas besoin de précisé le type (**: :geometry**) puisque la colonne the_geom est de type geometry.

2.5.13 Question : Quelles sont les aires et les périmètres des bâtiments ?

La requête suivante

```
select data as batiment,
       cast(area2d(the_geom) as decimal(15,2))||' m carre' as Aire,
       cast(perimeter(the_geom) as decimal(15,2))||' m' as Perimetre
          from buildings
```

nous renverra comme réponse

```
          batiment           |    aire          | perimetre
-----------------------------+------------------+-----------
 Collège Arthur Rimbaud      | 1370.00 m carre | 187.61 m
 Résidence des Mousquetaires | 2915.00 m carre | 216.00 m
 Hotel des Impots            | 1300.00 m carre | 190.00 m
 E.D.F                       | 476.43 m carre  | 88.54 m
 Bibliothèque Victor Hugo    | 900.00 m carre  | 176.57 m
 Mairie                      | 936.00 m carre  | 141.70 m
 Office du Tourisme          | 205.55 m carre  | 58.92 m
(7 rows)
```

2.5.14 Question : Qui est dans le bâtiment Résidence des Mousquetaires ?

La requête suivante

```
 select personnes.data as personnes_dans_batiment_2 from personnes,buildings
     where within(personnes.the_geom,buildings.the_geom)
          and buildings.data = 'Résidence des Mousquetaires';
```

nous renverra comme réponse

```
 personnes_dans_batiment_2
---------------------------
 personne 1
 personne 2
 personne 3
 personne 4
(4 rows)
```

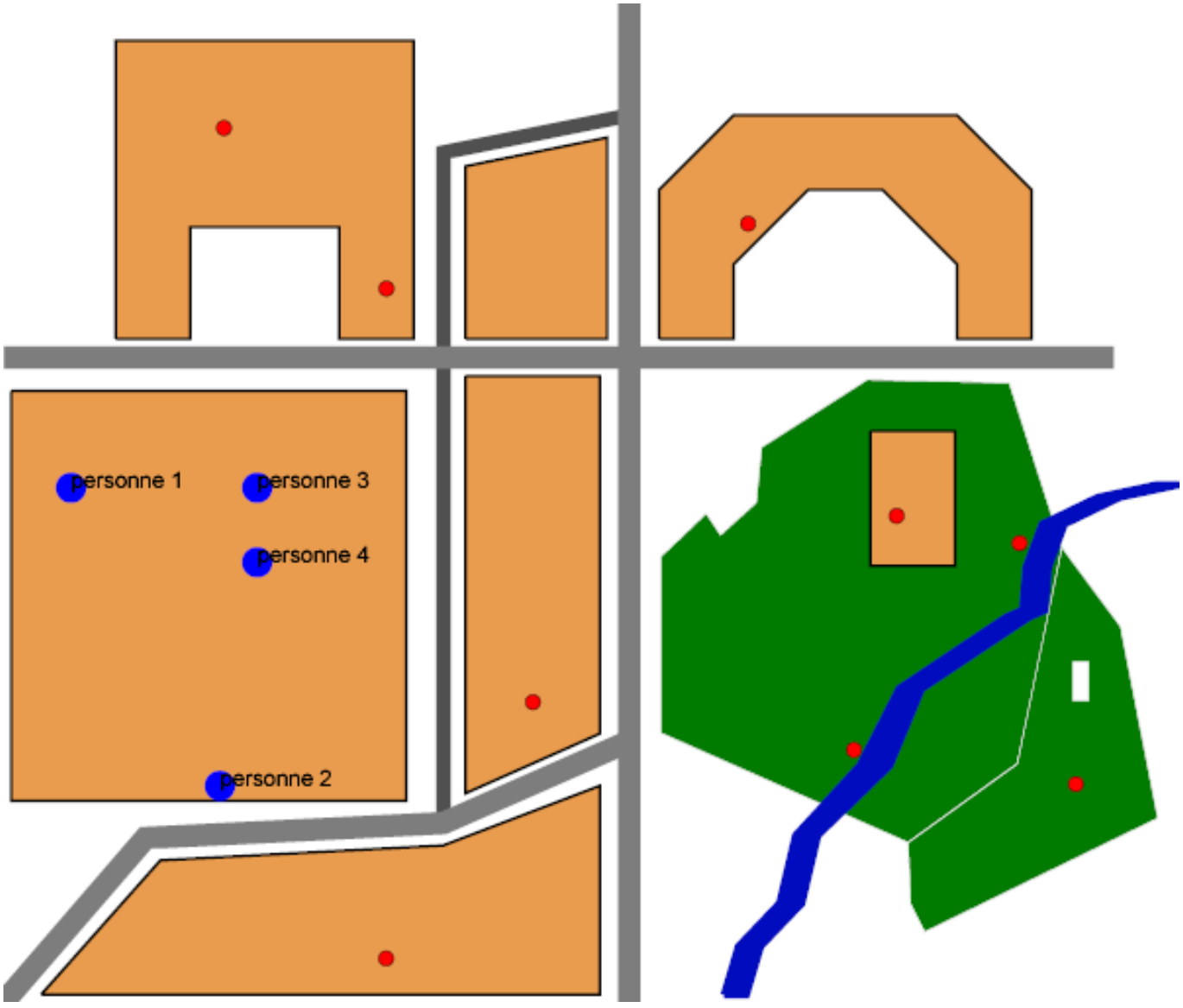

FIG. 2.8 – Personnes présentes dans le bâtiment Résidence des Mousquetaires

2.5.15 Question : Quelles distances séparent les bâtiments ?

La requête suivante

```
select h.data as batiment_1,t.data as batiment_2,
cast(Distance(t.the_geom,h.the_geom) as decimal(15,2))||' m'
```

```
 as distance_entre_batiment from buildings t, buildings h where h.data!=t.data;
```

nous renverra comme réponse

```
         batiment_1          |         batiment_2          | distance_entre_batiment
-----------------------------+-----------------------------+-------------------------
 Résidence des Mousquetaires | Collège Arthur Rimbaud      | 6.25 m
 Hotel des Impots            | Collège Arthur Rimbaud      | 64.97 m
 E.D.F                       | Collège Arthur Rimbaud      | 60.00 m
 Bibliothèque Victor Hugo    | Collège Arthur Rimbaud      | 60.53 m
 Mairie                      | Collège Arthur Rimbaud      | 5.47 m
 Office du Tourisme          | Collège Arthur Rimbaud      | 46.82 m
 Collège Arthur Rimbaud      | Résidence des Mousquetaires | 6.25 m
 Hotel des Impots            | Résidence des Mousquetaires | 7.00 m
 E.D.F                       | Résidence des Mousquetaires | 10.63 m
 Bibliothèque Victor Hugo    | Résidence des Mousquetaires | 34.71 m
 Mairie                      | Résidence des Mousquetaires | 8.00 m
 Office du Tourisme          | Résidence des Mousquetaires | 62.36 m
 Collège Arthur Rimbaud      | Hotel des Impots            | 64.97 m
 Résidence des Mousquetaires | Hotel des Impots            | 7.00 m
 E.D.F                       | Hotel des Impots            | 7.00 m
 Bibliothèque Victor Hugo    | Hotel des Impots            | 33.00 m
 Mairie                      | Hotel des Impots            | 8.60 m
 Office du Tourisme          | Hotel des Impots            | 62.60 m
 Collège Arthur Rimbaud      | E.D.F                       | 60.00 m
 Résidence des Mousquetaires | E.D.F                       | 10.63 m
 Hotel des Impots            | E.D.F                       | 7.00 m
 Bibliothèque Victor Hugo    | E.D.F                       | 7.00 m
 Mairie                      | E.D.F                       | 5.00 m
 Office du Tourisme          | E.D.F                       | 37.47 m
 Collège Arthur Rimbaud      | Bibliothèque Victor Hugo    | 60.53 m
 Résidence des Mousquetaires | Bibliothèque Victor Hugo    | 34.71 m
 Hotel des Impots            | Bibliothèque Victor Hugo    | 33.00 m
 E.D.F                       | Bibliothèque Victor Hugo    | 7.00 m
 Mairie                      | Bibliothèque Victor Hugo    | 9.43 m
 Office du Tourisme          | Bibliothèque Victor Hugo    | 12.40 m
 Collège Arthur Rimbaud      | Mairie                      | 5.47 m
 Résidence des Mousquetaires | Mairie                      | 8.00 m
 Hotel des Impots            | Mairie                      | 8.60 m
 E.D.F                       | Mairie                      | 5.00 m
 Bibliothèque Victor Hugo    | Mairie                      | 9.43 m
 Office du Tourisme          | Mairie                      | 36.36 m
 Collège Arthur Rimbaud      | Office du Tourisme          | 46.82 m
 Résidence des Mousquetaires | Office du Tourisme          | 62.36 m
 Hotel des Impots            | Office du Tourisme          | 62.60 m
 E.D.F                       | Office du Tourisme          | 37.47 m
 Bibliothèque Victor Hugo    | Office du Tourisme          | 12.40 m
 Mairie                      | Office du Tourisme          | 36.36 m
(42 rows)
```

2.5.16 Question : Combien de points composent chaque objet de la table great_roads ? - NumPoints() -

Nous aurons ici recours à la fonction `NumPoint()` qui permet de détermimer le nombre de points qui compose un objet :

```
SELECT data,AsText(the_geom),NumPoints(the_geom) FROM great_roads
```

nous renvoit effectivement

```
           data           |              astext                | numpoints
--------------------------+------------------------------------+-----------
 Rue Paul Valéry          | LINESTRING(1 1,20 23,60 25,85 36)  |         4
 Rue du Général de Gaulle | LINESTRING(1 87.5,150 87.5)        |         2
 Rue Aristide Briand      | LINESTRING(85 1,85 135)            |         2
(3 lignes)
```

2.5.17 Question : Dans la table great_roads, quels sont les premier et dernier point de la Rue Paul Valéry ? - StartPoint(), EndPoint() -

`StartPoint()` et `EndPoint()` permettent facilement de répondre à cette question. D'où la requête suivante

```
SELECT data,AsText(the_geom),AsText(StartPoint(the_geom)),AsText(EndPoint(the_geom))
FROM great_roads where data like '%Valéry%'
```

```
      data       |              astext                |   astext    |    astext
-----------------+------------------------------------+-------------+--------------
 Rue Paul Valéry | LINESTRING(1 1,20 23,60 25,85 36)  | POINT(1 1)  | POINT(85 36)
(1 ligne)
```

2.5.18 Question : Quels sont les points d'intersection entre les petites routes (small_roads) et les grandes routes (great_roads) ?

On pourra utiliser les fonctions respectives AsText() pour avoir les points en WKT et Intersects() pour les tests d'intersection

```
SELECT s.data,g.data,AsText(Intersection(s.the_geom,g.the_geom))
FROM small_roads s, great_roads g
WHERE Intersects(s.the_geom,g.the_geom)
```

renvoyant comme résultat

```
     data     |           data           |     astext
--------------+--------------------------+----------------
 Rue Figaro   | Rue du Général de Gaulle | POINT(60 87.5)
 Rue Figaro   | Rue Aristide Briand      | POINT(85 120)
 Rue Voltaire | Rue Paul Valéry          | POINT(60 25)
 Rue Voltaire | Rue du Général de Gaulle | POINT(60 87.5)
(4 lignes)
```

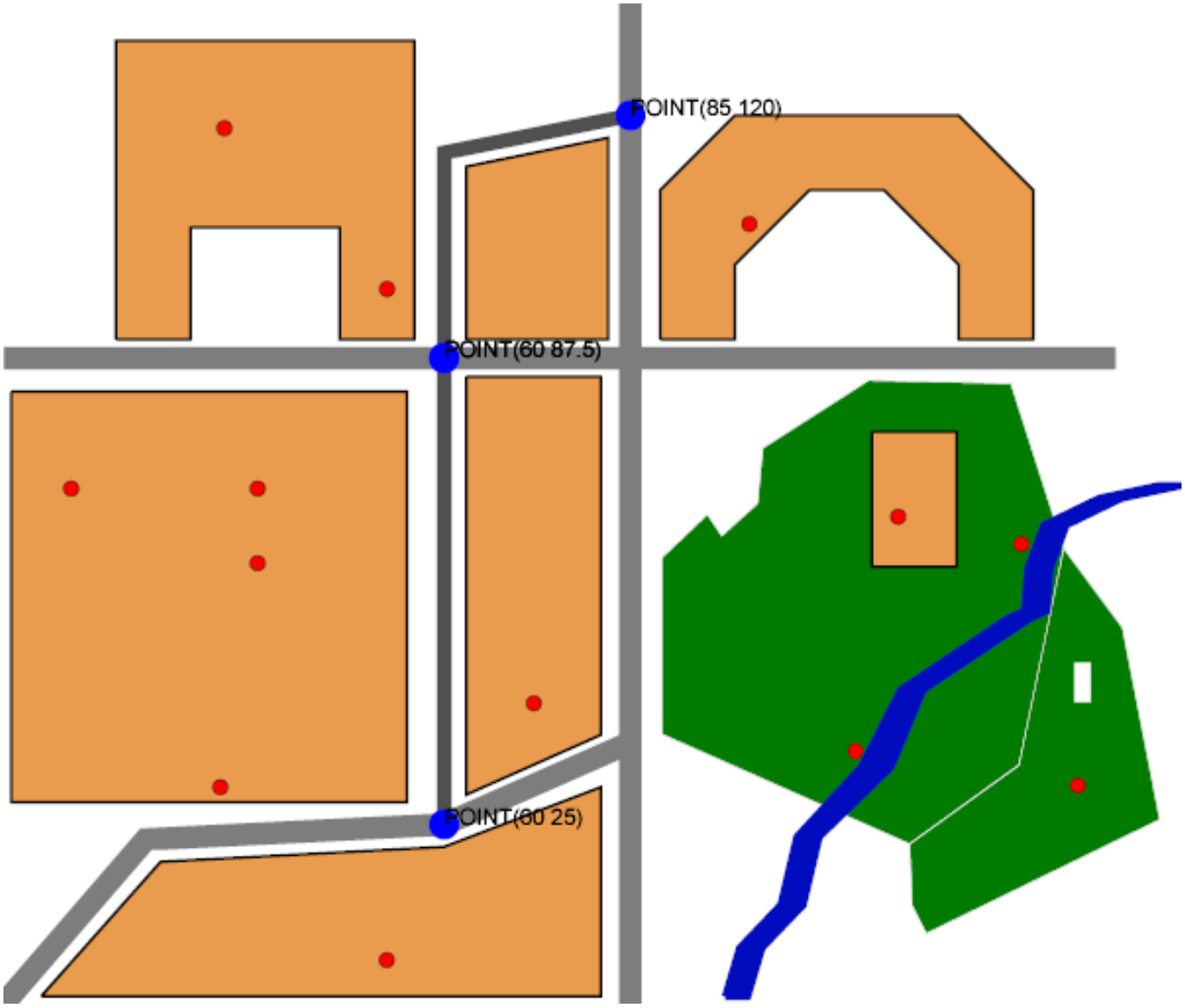

FIG. 2.9 – Points d'intersection entre les tables small_roads et great_roads

2.5.19 Question : Quel bâtiment est le plus proche de la personne 2 ?

Plusieurs formulations sont possibles. Les requêtes suivantes

```
select q.data from personnes z,buildings q
where z.data like 'personne 2' and
Distance(z.the_geom,q.the_geom)=
(select min(foo.valeur) from (select h.data as nom,distance(t.the_geom,h.the_geom)
as valeur from personnes t, buildings h where t.data like 'personne 2') foo);
```

et celle-ci

```
SELECT data from(
SELECT a.data,Distance(a.the_geom,b.the_geom) FROM buildings a,personnes b
  WHERE b.data='personne 2'
   ORDER BY  distance ASC limit 1)
```

```
as foo;
```

nous renvoient comme réponse

```
            data
-----------------------------
 Résidence des Mousquetaires
(1 ligne)
```

2.5.20 Question : Quel bâtiment ne contient aucune personne ?

La requête suivante

```
select b.data from buildings b,
                  (select geomunion(the_geom) from personnes) gp
                where
                    not intersects(gp.geomunion,b.the_geom);
```

nous renverra comme réponse

```
 data
-------
 E.D.F
(1 row)
```

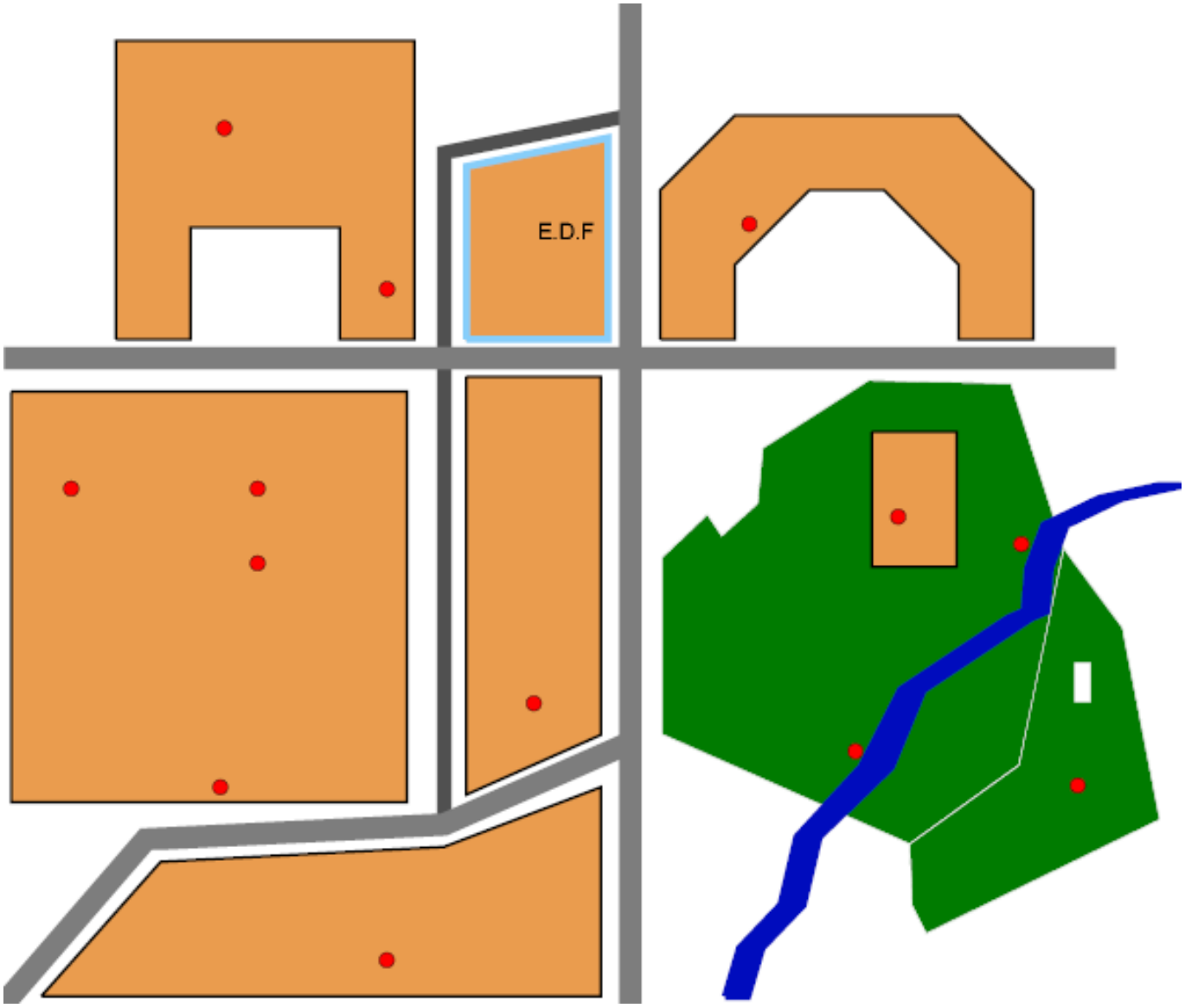

FIG. 2.10 – Bâtiment ne contenant aucune personne : EDF

2.5.21 Question : Quels sont les personnes présentes dans les bâtiments ?

On utilisera pour cela la requête suivante

```
select p.data,b.data from personnes p, buildings b where intersects(p.the_geom,b.the_geom)
```

qui nous renverra

```
     data     |            data
--------------+-----------------------------
 personne 1   | Résidence des Mousquetaires
 personne 2   | Résidence des Mousquetaires
 personne 3   | Résidence des Mousquetaires
 personne 4   | Résidence des Mousquetaires
 personne 5   | Hotel des Impots
 personne 6   | Hotel des Impots
```

```
 personne 7  | Bibliothèque Victor Hugo
 personne 9  | Office du Tourisme
 personne 12 | Mairie
 personne 13 | Collège Arthur Rimbaud
(10 lignes)
```

2.5.22 Question : Combien y-a-t-il de personnes par bâtiments ?

Cette requête est un peu plus poussée que la précédente car il se peut que ce nombre puisse valoir zéro. On peut par exemple utiliser la négation de la fonction de Contains() pour obtenir le nombre de personnes non présentes par bâtiment. Ce dernier nombre est alors soustrait du nombre total de personnes - nombre que l'on peut obtenir par le fonction Count() de PostgreSQL. Ce qui donnerait par exemple comme requête

```
SELECT (SELECT Count(*) FROM personnes)-Count(p.data) AS nombre_de_personnes,b.data as  ↩
    batiment
FROM
 buildings b , personnes p
WHERE NOT Contains(b.the_geom,p.the_geom)
GROUP BY b.data ORDER BY b.data
```

nous renvoyant comme résultat

```
 nombre_de_personnes |          batiment
---------------------+----------------------------
                   1 | Bibliothèque Victor Hugo
                   1 | Collège Arthur Rimbaud
                   0 | E.D.F
                   2 | Hotel des Impots
                   1 | Mairie
                   1 | Office du Tourisme
                   4 | Résidence des Mousquetaires
(7 lignes)
```

2.5.23 Question : Quel est l'aire d'intersection entre la rivière et les parcs ?

La requête suivante

```
 select cast(area2d(intersection(r.the_geom,p.the_geom)) as decimal(15,1)) ||' m carre' as  ↩
     Aire
                     from rivers r,
                           parcs p
                    where intersects(r.the_geom,p.the_geom);
```

nous renverra comme réponse

```
   aire
----------
 123.1 m carre
(1 ligne)
```

L'affichage avec MapServer peut être obtenu en utilisant par exemple le LAYER suivant dans une mapfile

```
  LAYER
    CONNECTION "user=david dbname=madatabase host=localhost"
    CONNECTIONTYPE POSTGIS
    DATA "intersection FROM (select r.gid,intersection(r.the_geom,p.the_geom) FROM rivers r ↩
        ,parcs p
 WHERE intersects(r.the_geom,p.the_geom)) foo USING UNIQUE gid USING SRID=-1"
```

```
    NAME "requete"
    TYPE POLYGON
    STATUS DEFAULT
    CLASS
      STYLE
        ANGLE 360
          OUTLINECOLOR 0 0 0
          COLOR 134 100 0
        SIZE 5
        SYMBOL 0
      END
    END
  END
```

Au niveau du visuel, on obtiendrait

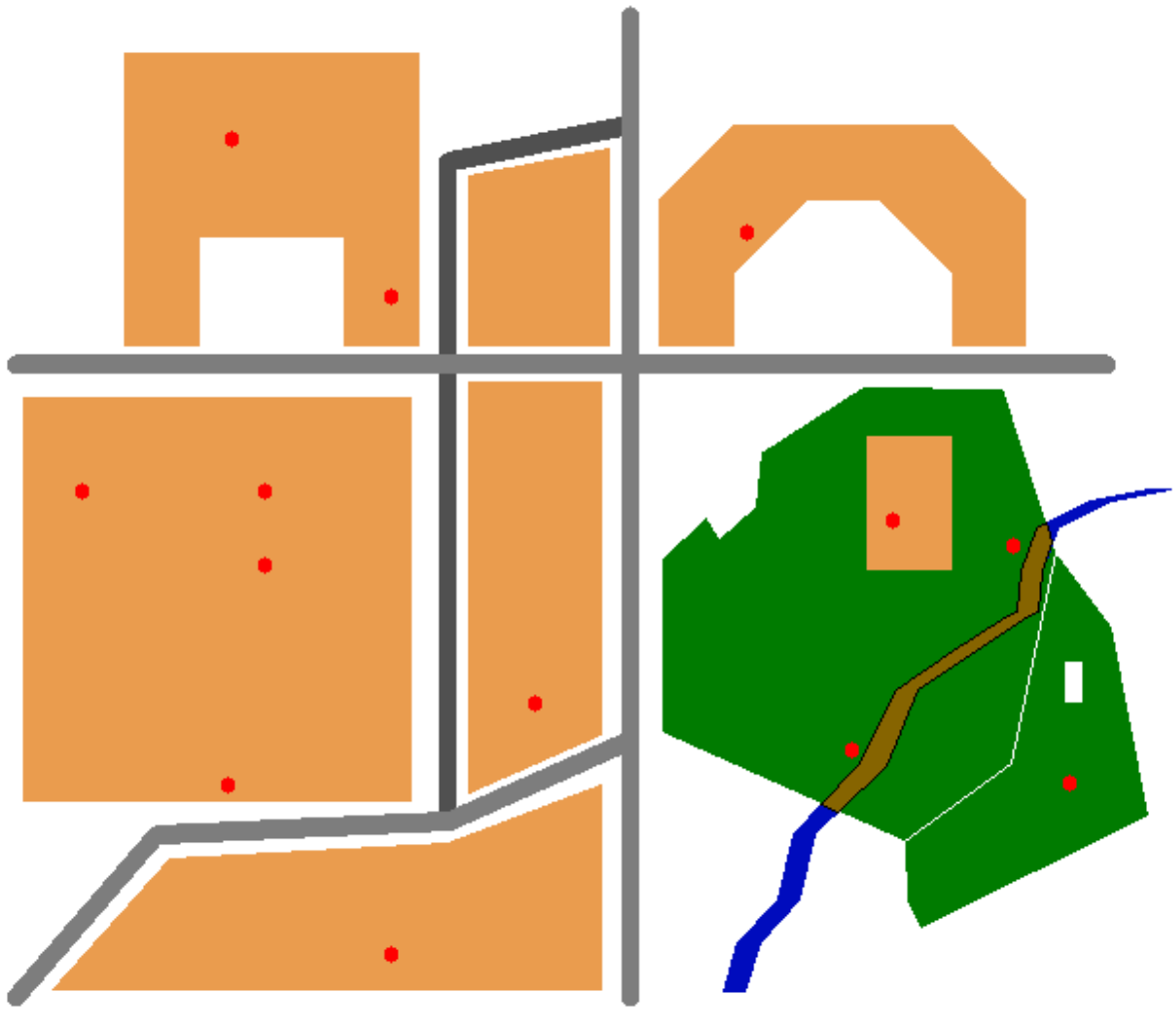

FIG. 2.11 – Intersection entre la rivière et les parcs

2.5.24 Question :Quel bâtiment est contenu dans le parc Mangueret I ? - Contains() -

La fonction `Contains(A,B)` permet de savoir si l'objet A contient l'objet B. De ce fait, la requête

```
select b.data from buildings b,parcs where
  Contains(parcs.the_geom,b.the_geom)
```

renvoit le résultat immédiat

```
        data
--------------------
 Office du Tourisme
(1 row)
```

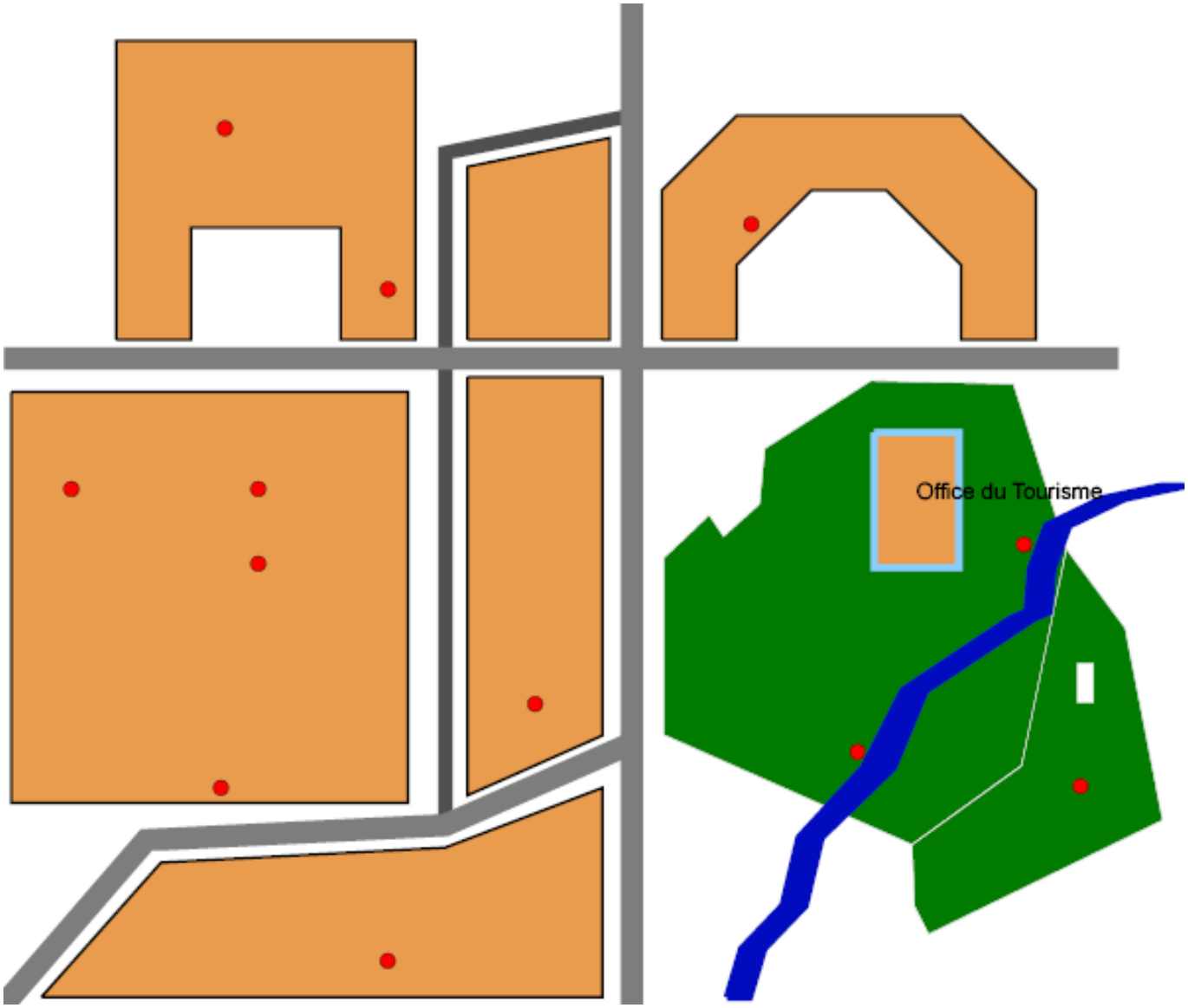

FIG. 2.12 – Bâtiment contenu dans le parc Mangueret I : Office du Tourisme

2.5.25 Question :Quelles sont les personnes proches de la rivière dans un rayon de 5 mètres ? - Buffer () -

Ici, nous allons utilisé un combinaison des fonctions Contains() que nous avons vu précédement et de aa fonction `Buffer(A,r)` où `A` est l'objet géométrique auquel on applique un buffer de rayon `r` . Pour rappel , Buffer(A,r) est l'objet - ensemble de points - dont la distance à l'objet A est inférieure où égale à r. La requête

```
select p.data from personnes p,rivers
where contains(buffer(rivers.the_geom,5),p.the_geom);
```

nous renvoit alors comme réponse

```
     data
---------------
 personne 8
 personne 10
(2 rows)
```

Au niveau de MapServer pour un rendu visuel, il faut deux layers pour obtenir les résultats souhaités. Il en faut un pour afficher le buffer en question et l'autre pour afficher les résultats de la requête précédente.

1. Pour afficher le buffer, on pourra utiliser le layer suivant :

```
LAYER
  CONNECTION "user=david dbname=madatabase host=localhost"
  CONNECTIONTYPE POSTGIS
  DATA "buffer from (select gid,buffer(rivers.the_geom,5) from rivers) foo
  USING UNIQUE gid USING SRID=-1"
  NAME "requete_1"
  TYPE LINE
  STATUS DEFAULT
  CLASS
    STYLE
      OUTLINECOLOR 20 156 78
      SIZE 10
      SYMBOL 0
    END
  END
END
```

2. Pour afficher les données attendues, on pourra proposer :

```
LAYER
  CONNECTION "user=david dbname=madatabase host=localhost"
  CONNECTIONTYPE POSTGIS
  DATA "the_geom from (select * from personnes where
  contains(buffer(rivers.the_geom,5),personnes.the_geom)) foo
  USING UNIQUE gid USING SRID=-1"
  NAME "requete_2"
  LABELITEM "data"
  STATUS DEFAULT
  TYPE POINT
  CLASS
   LABEL
      COLOR 22 8 3
    END
    STYLE
      COLOR 255 0 0
      SIZE 8
      SYMBOL "circle"
    END
  END
END
```

NOTE
Il est aussi possible de n'avoir recours qu'à un seul layer pour afficher à la fois l'objet et les personnes attendues ! Ce sera le cas lors de la prochaine section - sur les requêtes concernant une certaine table `communes_avoisinantes` -.

Le visuel sera alors le suivant

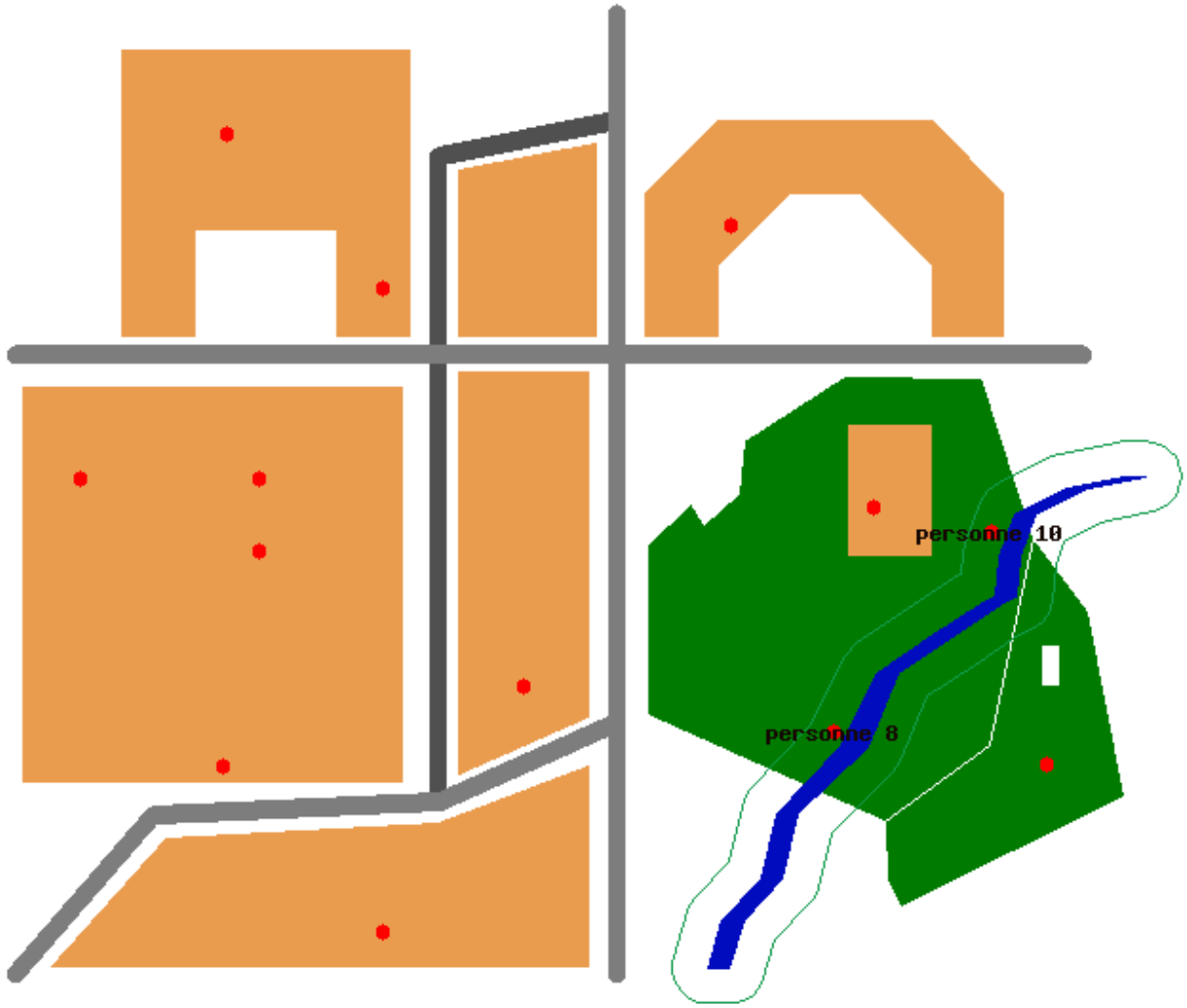

FIG. 2.13 – Buffer de 5 mètres sur la rivière : les personnes 8 et 10 y sont présentes

2.5.26 Question : Quel parc contient un "trou" ? - Nrings() -

La requête suivante avec l'emploi de la fonction `Nrings(geometry)`

```
select data from parcs where nrings(the_geom)>1;
```

nous renvoit comme réponse

```
       data
--------------------
```

```
 Parc Mangueret II
(1 ligne)
```

En effet, cette fonction permet par exemple dans le cas d'un POLYGON ou d'un MULTIPOLYGON de déterminer le nombre de polygones faisant partie de la collection constituant l'objet :

```
select data,nrings(the_geom) from parcs;
```

```
       data         | nrings
--------------------+--------
 Parc Mangueret I   |      1
 Parc Mangueret II  |      2
(2 lignes)
```

2.5.27 Question : Quels sont les bâtiments que rencontrent la ligne qui relie la personne 5 à la personne 13 ? - MakeLine() -

Pour répondre à cette question, il faut commencer par construire cette ligne en utilisant la fonction MakeLine(geometry,geometry) qui permet de relier entre eux deux points :

```
select astext(makeline(a.the_geom,b.the_geom)) from personnes a,personnes b
where a.data='personne 5' and b.data='personne 13';
```

```
               astext
-------------------------------------
 LINESTRING(30.54 118.28,52.32 6.84)
(1 ligne)
```

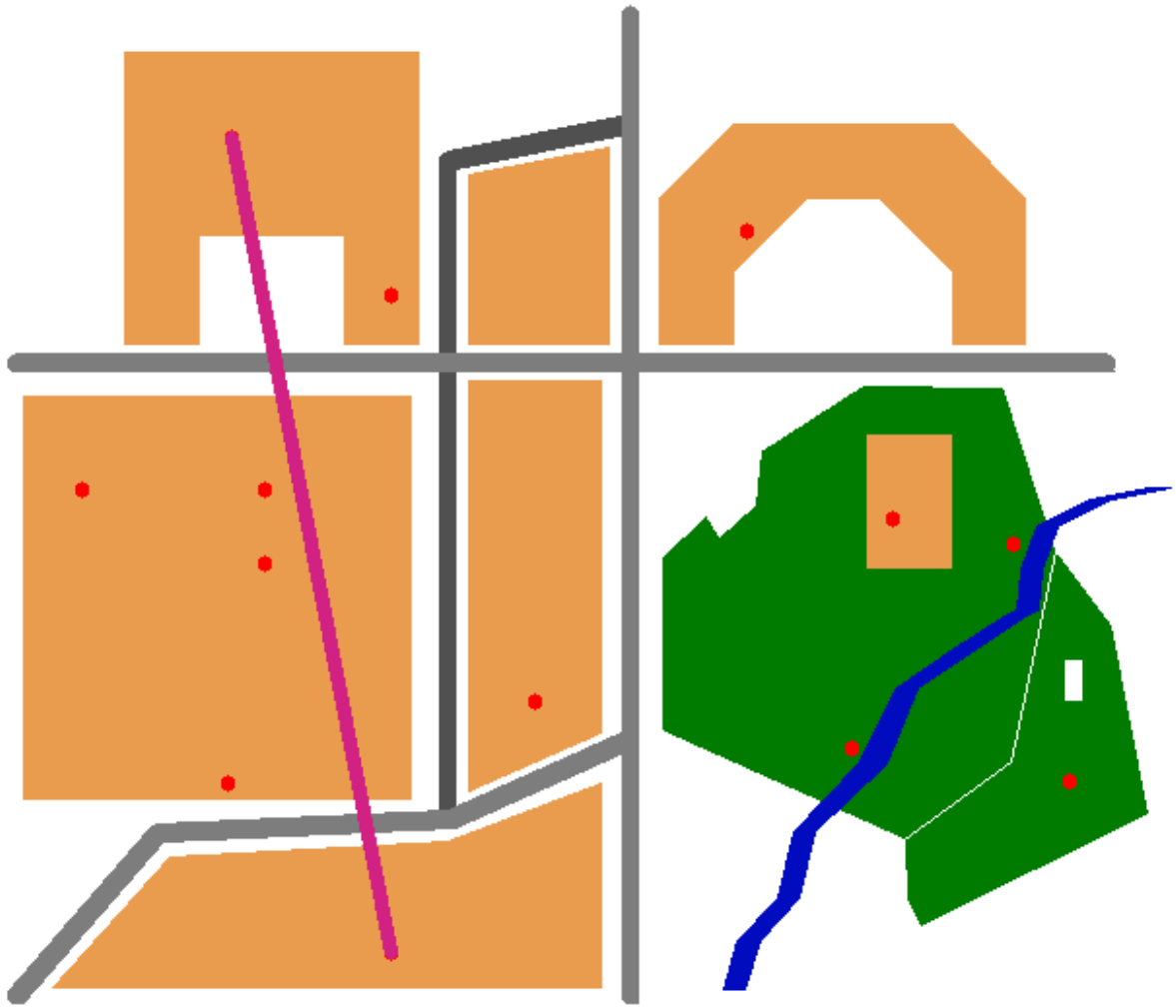

FIG. 2.14 – Ligne reliant les points désignant les personnes 5 et 13

Il en résultera donc la requête suivante

```
select build.data from buildings build where
intersects(
build.the_geom,
(select makeline(a.the_geom,b.the_geom) from personnes a,personnes b where a.data='personne ↩
    5'
and b.data='personne 13')
);
```

```
           data
----------------------------
 Collège Arthur Rimbaud
 Résidence des Mousquetaires
 Hotel des Impots
(3 lignes)
```

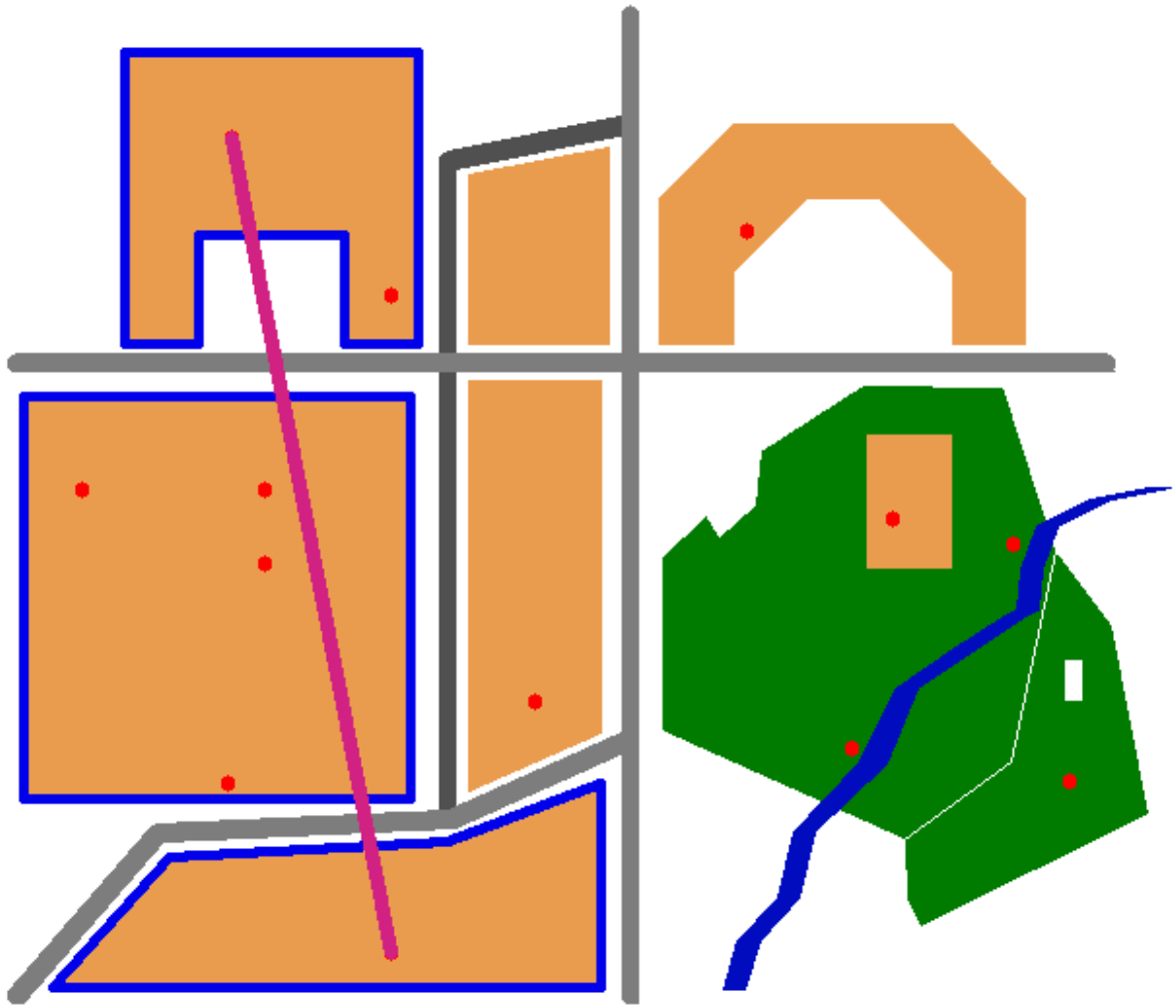

FIG. 2.15 – Bâtiments (table buildings) que rencontre la ligne reliant les points désignant les personnes 5 et 13

2.5.28 Application : Utiliser les déclencheurs (triggers) en PL/PGSQL de PostgreSQL pour suivre à la trace la personne 7 quand elle se déplace. Selon sa position, savoir quel est le bâtiment qui lui est le plus proche ou le bâtiment dans lequel elle se trouve ?

2.5.28.1 Fonction, table et trigger nécessaires

Le language PL/PGSQL sert tirer profit des procédures déclencheurs. Nous allons ici créer une table `suivi_de_la_personne_7` qui enregistrera à tout moment demandé enregistrera les informations suivantes :

utilisateur correspondant au nom de l'utilsateur sous de PostgreSQL qui aura fait un update sur les données spatiales de la "personne 7" ;

date correspondant à l'heure à laquelle a eu lieu le déplacement ;

suivi qui précisera si la personne est "dans" ou "proche" du batiment ;

position qui donnera la position de la personne.

Nous aurons aussi besoin d'une fonction `get_info_on_personne_7()` écrite en PL/PGSQL appelé par le déclencheur qui nous renverra le bâtiment qui contient la personne 7 et dans le cas contraire le premier bâtiment qui lui est le plus proche.

```
/*
    Rappel pour effacer un trigger
    drop trigger suivi_7_date on personnes;
*/

/*
   Table qui contiendra le suivi de la
   personne 7 lors de ses déplacements
  avec effacement éventuel avant création
*/
select drop_table_if_exists('suivi_de_la_personne_7',false);

create table suivi_de_la_personne_7(
utilisateur text,
date text,
suivi text,
position text
);
/*
   Fonction appelée par le trigger
*/
create or replace function get_info_on_personne_7() returns trigger as $$
declare
j record;
begin
    if (TG_OP = 'UPDATE') then
        SELECT into j data,distance,astext(the_geom)  from(
          SELECT a.data,Distance(a.the_geom,b.the_geom),b.the_geom FROM buildings a, ↩
              personnes b
              WHERE b.data='personne 7'
            ORDER BY  distance ASC limit 1)
           as foo;
           if j.distance=0 then
           insert into suivi_de_la_personne_7 values
          (current_user,now(),'La personne 7 est dans le batiment '''||j.data::text||'''',j ↩
              .astext::text);

                        RAISE NOTICE 'La personne 7 est dans le batiment ''%''',j.data;
                        else
                        insert into suivi_de_la_personne_7 values
                       (current_user,now(),'La personne 7 est proche du batiment '''||j. ↩
                           data::text||'''',j.astext::text);

                        RAISE NOTICE 'La personne 7 est proche du batiment ''%''',j.data;

                        end if;
    end if;
    return new;
end;

$$ language plpgsql;
/*
   Création du trigger
*/
CREATE TRIGGER suivi_7_date AFTER  UPDATE ON personnes
    EXECUTE PROCEDURE get_info_on_personne_7();
```

2.5.28.2 Modifications de la position par la commande UPDATE et GeometryFromText

Procédons maintenant à quelques exemples de mise à jour des données spatiales de la personne 7 :

```
madatabase=# update personnes set the_geom=geometryfromtext('POINT(100.94 105.44)',-1)  ↩
    where data='personne 7';
INFO:  La personne 7 est dans le batiment 'Bibliothèque Victor Hugo'
UPDATE 1
madatabase=#  update personnes set the_geom=geometryfromtext('POINT(100 70)',-1) where data ↩
    ='personne 7';
INFO:  La personne 7 est proche du batiment 'Office du Tourisme'
UPDATE 1
madatabase=# update personnes set the_geom=geometryfromtext('POINT(120 66)',-1) where data ↩
    ='personne 7';
INFO:  La personne 7 est dans le batiment 'Office du Tourisme'
UPDATE 1
madatabase=#  update personnes set the_geom=geometryfromtext('POINT(0 0)',-1) where data=' ↩
    personne 7';
INFO:  La personne 7 est proche du batiment 'Collège Arthur Rimbaud'
UPDATE 1
madatabase=# update personnes set the_geom=geometryfromtext('POINT(60 87.5)',-1) where data ↩
    ='personne 7';
INFO:  La personne 7 est proche du batiment 'E.D.F'
UPDATE 1
```

2.5.28.3 Suivi des déplacements

La simple requête suivante nous renvoit les divers informations obtenues lors du déplacement de la personne 7

```
testgis=# select * from suivi_de_la_personne_7 ;
 utilisateur |             date              |                                suivi  ↩
                                |       position
-------------+-------------------------------+-------------------------------------------------------

 postgres    | 2006-01-12 12:05:01.989762+01 | La personne 7 est dans le batiment ' ↩
     Bibliothèque Victor Hugo' | POINT(100.94 105.44)
 postgres    | 2006-01-12 12:05:20.824226+01 | La personne 7 est proche du batiment 'Office ↩
      du Tourisme'     | POINT(100 70)
 postgres    | 2006-01-12 12:05:41.106236+01 | La personne 7 est dans le batiment 'Office  ↩
     du Tourisme'       | POINT(120 66)
 postgres    | 2006-01-12 12:06:05.024382+01 | La personne 7 est proche du batiment ' ↩
     Collège Arthur Rimbaud' | POINT(0 0)
 postgres    | 2006-01-12 12:07:31.859971+01 | La personne 7 est proche du batiment 'E.D.F' ↩
                       | POINT(60 87.5)
(6 lignes)
```

C'est le genre de petite application qui peut par exemple servir lors de suivi par une application-cliente (front-end) par le Web par exemple : interface cartographique en SVG ....

2.6 Cas pratique avec MapServer

Nous allons exposer ici des exemples de requêtes sur des données SIG réelles. Les situations citées ici sont tirées de notre travail quotidien en tant qu'utiisateur de PostGIS et de MapServer.

2.6.1 Importation des communes du Languedoc-Roussillon

Ici les communes de cette région sont "fournies" au format ESRI Shapefiles dans le fichier `communes_lr.shp`. Ces données sont déjà identifiées dans le système de projection français Lambert II Carto Etendu. Sachant que dans ce cas précis, l'identifiant de

projection (srid) vaut 27582, pour les importer dans notre base madatabase, il nous suffira de faire

```
shp2pgsql -s 27582 -DI communes_lr.shp communes_lr | psql madatabase
```

Reportez-vous à la documentation de shp2pgsql pour les options ici utilisées. Nous noterons au passage que la région du Languedoc-Roussillon est composée de 1545 communes soit 1545 enregistrements dans notre table.

2.6.2 Afficher les informations relatives au Lambert II Etendu depuis la table spatial_ref_sys

Pour avoir le maximum d'information sur la structure de cette table, je ne saurais vous conseiller la documentation en ligne sur le site de PostGIS. Comme dit dans ce document , c'est la table **spatial_ref_sys** de PostGIS qui contient les informations relatives aux divers systèmes de projection - table requise selon les spécifications de l'O.G.C (Open GIS Consortium). Excecutez la requête suivante par exemple pour voir les divers types de projections relatives à la France.

```
select srid,auth_name,auth_srid,srtext from spatial_ref_sys
where srtext like '%France%'
```

La ligne qui nous intéresse est celle pour laquelle srid vaut 27582. Son champs srtext vaut

```
PROJCS["NTF (Paris) / France II",GEOGCS["NTF (Paris)",asting",600000](...)AUTHORITY["EPSG ↩
    ","27582"]]
```

2.6.3 Question : Comment faire si on a oublié de préciser l'identifiant de système de projection ? - UpdateGeometrySrid() -

Dans le cas où les données ont déjà été importées et que lors de l'utilisation de shp2pgsql on a oublié d'utiliser l'option -s pour préciser le système de projection, PostGIS considère qu'il s'agit de données plannes du plan orthornormal. Il précise alors par défaut srid=-1. Ce qui peut-être contraignant par exemple si par la suite, nous soyons amenés à utiliser les fonctions Distance() ou Area2f() qui selon le système de projection peuvent donner des résultats faussées.

Pour préciser le système de projection, deux solutions possibles :

`solution 1` : recharger à nouveau les données en utilisant shp2pgsql avec les options respectives -s pour l'identifiant de projection et l'option -d qui permet d'effacer la table avant de recharger le fichier .shp d'origine. La commande sera donc

```
shp2pgsql -s 27582 -dDI communes_lr.shp communes_lr | psql madatabase
```

`solution 2` : garder les données déjà importées et modifier le srid à la volée. Pour cela, il faut avoir recours à la fonction UpdateGeometrySrid dont la synthaxe dans notre cas sera

```
SELECT UpdateGeometrySrid('communes_lr','the_geom',27582);
```

Dans les deux cas, les données sont mises à jours ainsi que les méta-données les concernant dans la table geometry_columns.

2.6.4 Création d'index spatiaux Gist, Vacuum de la base

La table que nous avons créée contient pas moins de 1545 enregistrements. Normalement la création d'index spatiaux a lieu en ayant recours par exemple

```
CREATE INDEX [index_spatial_table] ON [table]
 USING gist([colonne_geometrique] gist_geometry_ops);
```

Or l'option -I de shp2pgsql permet de créer automatiquement un index spatial - suffixé _the_geom_gist - en pour chacun de ces enregistrements et évite d'avoir à se soucier de leur création par l'emploi de cette requête. Par exemple ici, cette option fait alors directement appel ici pour la table communes_lr à la requête d'indexation spatiale suivante

```
CREATE INDEX communes_lr_the_geom_gist ON communes_lr
 USING gist(the_geom gist_geometry_ops);
```

La création des index spatiaux peut s'avérer "gourmand". Pour que le planificateur de requêtes de PostgreSQL dispose des informations statistiques nécessaires et adéquates pour savoir quand avoir recours aux index spatiaux selon la fonction spatial demandée, il est nécessaire parfois de faire un VACUUM ANALYZE sur la base en cours

Nous reviendrons sur cet aspect plus tard. Pour résumer, nous dirons juste que à la suite d'une ou plusieurs importation(s) de données importante pour lesquelles il y a eu besoin de création d'index spatiaux Gist, il est recommandé d'utiliser la requête VACUUM ANALYZE

```
VACUUM ANALYZE
```

NOTE
Pour PostgreSQL 8.x.x, cette commande permet de collecter les informations statistiques de l'emploi des index spatiaux

2.6.5 Question : Qu'elle est l'étendue géographique de la table communes_lr ? - Extent() -

Par définiton, je rappelle que l'étendue géographique d'un objet spatial dans une base de données est la plus petit fenêtre (=rectangle) qui le contienne. Ce dernier est défini par le quadruplet (Xmin,Ymin,Xmax,Ymax) . (Xmin,Ymin) (respectivement (Xmax,Ymax)) est le sommet inférieur gauche (respectivement le sommet supérieur droit) du rectangle dans tout système de projection orienté de gauche à droite horizontalement et de bas en haut vertivalement.

Pour connâitre ce quadruplet, la requête suivante

```
SELECT Xmin(foo.extent),
       Ymin(foo.extent),
       Xmax(foo.extent),
       Ymax(foo.extent)
 FROM
      (SELECT Extent(the_geom) FROM communes_lr) AS foo
```

nous renvoit comme résultat

```
  xmin   |    ymin     |  xmax  |    ymax
---------+-------------+--------+-------------
 547293 | 1703209.875 | 801423 | 1997767.125
(1 row)
```

Ce dernier implique donc que tous les objets de la colonne géométrique the_geom sont contenus dans le quadruplet ci-dessus obtenu.

NOTE
Connaître l'étendue géographique peut s'avérer utile, notamment avec Mapserver dont les fichiers mapfiles possèdent un paramètre EXTENT qui correspond à l'étendue géogpraphique. Ici il s'agira de fournir ce quadruplet.

2.6.6 Visualisation des données avec MapServer

Cette section suppose que l'emploi de MapServer vous soit familier. Je ne passerais pas ici en revue les divers fonctionnements de MapServer. Je profite de l'occasion juste pour détailler de quoi est composer la plus petite couche - LAYER au sens de MapServer - pour PostGIS. Je pars ici du principe que vous avez Apache, Php et PhpMapScript d'installés sur votre machine.

NOTE
Pour disposer de tous ces outils, le mieux est de recourir au paquet MS4W (MapServer For Windows) disponible sur le site http ://www.maptools.org

Pour construire mon image sur les communes du Languedoc-Roussillon, le premier paramètre à fournir est bien l'EXTENT. Or selon une des dimensions de l'image que je me fixe (longueur ou hauteur), il me faut aussi savoir qu'elle sera la valeur de l'autre dimensions.

2.6.6.1 Pour une image de longueur 500 pixels, quelle doit être la hauteur de l'image en fonction de l'étendue géographique ?

Il suffit pour cela d'effectuer une simple règle de trois que me donne la requête suivante inspirée de la requête de la section précédente :

```
SELECT
       500*(abs(Ymax(foo.extent)-Ymin(foo.extent))/abs(Xmax(foo.extent)-Xmin(foo.extent)))
       AS hauteur
 FROM (SELECT Extent(the_geom) FROM communes_lr) AS foo
```

qui me renvoie

```
     hauteur
------------------
 579.540491087239
(1 ligne)
```

2.6.6.2 Mapfile et Script PHP

Pour afficher ma table communes_lr, ayant l'étendue géographique et la hauteur de l'image, je peux par exemple avoir comme mapfile appelée `communes_lr.map`

```
MAP
  EXTENT 547293 1703209.875 801423 1997767.125
  IMAGECOLOR 255 255 255
  IMAGETYPE png
  SIZE 500 579.540491087239

  WEB
    IMAGEPATH "c:/wamp/www/tutorial/tmp/"
    IMAGEURL "/tutorial/tmp/"
  END
  #
  # Couche des communes
  #
  LAYER
    NAME "communes_lr"
    CONNECTION "user=david dbname=madatabase host=localhost"
    CONNECTIONTYPE POSTGIS
    DATA "the_geom from communes_lr"
    STATUS DEFAULT
    TYPE POLYGON
    CLASS
      STYLE
          COLOR 255 255 255
          OUTLINECOLOR 0 0 0
      END
    END
  END

END
```

avec comme paramètres au niveau générale

EXTENT qui prend comme paramètres le quadruplet Xmin Ymin Xmax Ymax ;

IMAGECOLOR prend comme valeur un RGB (Red Green Blue) pour définir le fond de couleur. Ici 255 255 255 pour avoir du blanc ;

IMAGETYPE png pour préciser que l'image produite sera au format png. cela suppose que MapServer est été compilé avec l'option --with-png pr défaut comme OUTPUTFORMAT ;

SIZE qui prend comme paramètres Longueur Hauteur

au niveau du WEB :

IMAGEPATH pour préciser le chemin d'accès absolue sur le disque où seront générées les images produites ;

IMAGEURL url relative à la valeur donnée par IMAGEPATH

au niveau de la couche - LAYER - pour la table communes_lr

CONNECTION suivi des paramètres de connection à PostgreSQL ;

CONNECTIONTYPE POSTGIS pour préciser qu'il s'agit d'une connection à une base de données PostGIS ;

DATA dont la pseudo-valeur doit être "colonne_geometrique from table". On met donc ici "the_geom from communes_lr" sans le mot-clé SELECT.

TYPE qui précise la nature géométrique des objets à afficher. TYPE peut prendre la valuer POINT, LINE ou POLYGON.

le reste au niveau du CLASS/STYLE concerne l'habillage pour l'affichage des objets. Ici un contour noir (OUTLINECOLOR 0 0 0 et un intérieur blanc (COLOR 255 255 255)

Au niveau de php grâce à - l'extension phpmapscript si vous l'avez installé - pour générer l'image on peut utiliser le script suivant -

```
<html>
<body>
<?php
    $sw_MapFile = "./mapfiles/communes_lr.map";

    $map = ms_newMapObj( $sw_MapFile );

    $image = $map->draw();
$image_url = $image->saveWebImage(MS_PNG,1,1,0);

echo "<IMG
BORDER=0
SRC='".$image_url."'
width='".$map->width."' height='".$map->height."'/>
<BR>";
?>
</body>
</html>
```

On obtiendra ainsi le visuel suivant

FIG. 2.16 – Affichage des communes du Languedoc-Roussillon (MapServer+PhpMapScript)

2.6.7 Question : Quelles sont les comunes avoisinantes de Montpellier ?, Utilité des index spatiaux - Distance(), && -

Dans le cas de certaines fonctions de PostGIS, parmi lesquelles nous pouvons citer Distance() ; WithIn(), Intersects() [...]. ,il est parfois nécessaire selon la résultat escompté de les coupler à **&&** := opérateur de chevauchement des étendues géographiques.

En effet, celui-ci sait tirer profit des index spatiaux - en interne de son implémentation, nous dirons qu'il les exploite-. Comme son nom l'indique, il teste si les étendues géographiques de deux objets géométriques A et B sechevauchent.

A titre d'exemple, pour avoir la liste des communes qui avoisines Montpellier, il s'agit simplement des communes dont la distance à Montpellier est nulle, ce qui sous-entend donc que leurs étendues géographiques se chevauchent- nous aurons recours à l'utilisation de l'opérateur && pour accélérer le temps nécessaire à la requête. D'où la requête suivante

```
SELECT b.nom  FROM communes_lr a, communes_lr b
WHERE
     a.nom LIKE 'MONTPELLIER'
AND
     b.nom NOT LIKE 'MONTPELLIER'
AND
    Distance(a.the_geom,b.the_geom)=0
AND
    a.the_geom && b.the_geom
ORDER BY b.nom
```

qui nous renvoit

```
           nom
--------------------------
 CASTELNAU-LE-LEZ
 CLAPIERS
 GRABELS
 JUVIGNAC
 LATTES
 LAVERUNE
 MAUGUIO
 MONTFERRIER-SUR-LEZ
 SAINT-AUNES
 SAINT-CLEMENT-DE-RIVIERE
 SAINT-JEAN-DE-VEDAS
(11 rows)
```

Sans l'utilisation ici de `a.the_geom && b.the_geom`, le temps demandé par la requête s'avère long - j'en témoigne pour 1544 enregistrements à tester ! -.

NOTE
Il est également possible d'utiliser une fonction plus performante que la focntion Distance avec la condition Distance(A,B)=0 . On aurait pû avoir recours à la fonction Touches(A,B) qui permet de savoir si A et B se touchent puisqu'ici nous savons que les communes ne se "touchent" que par leur frontière

---

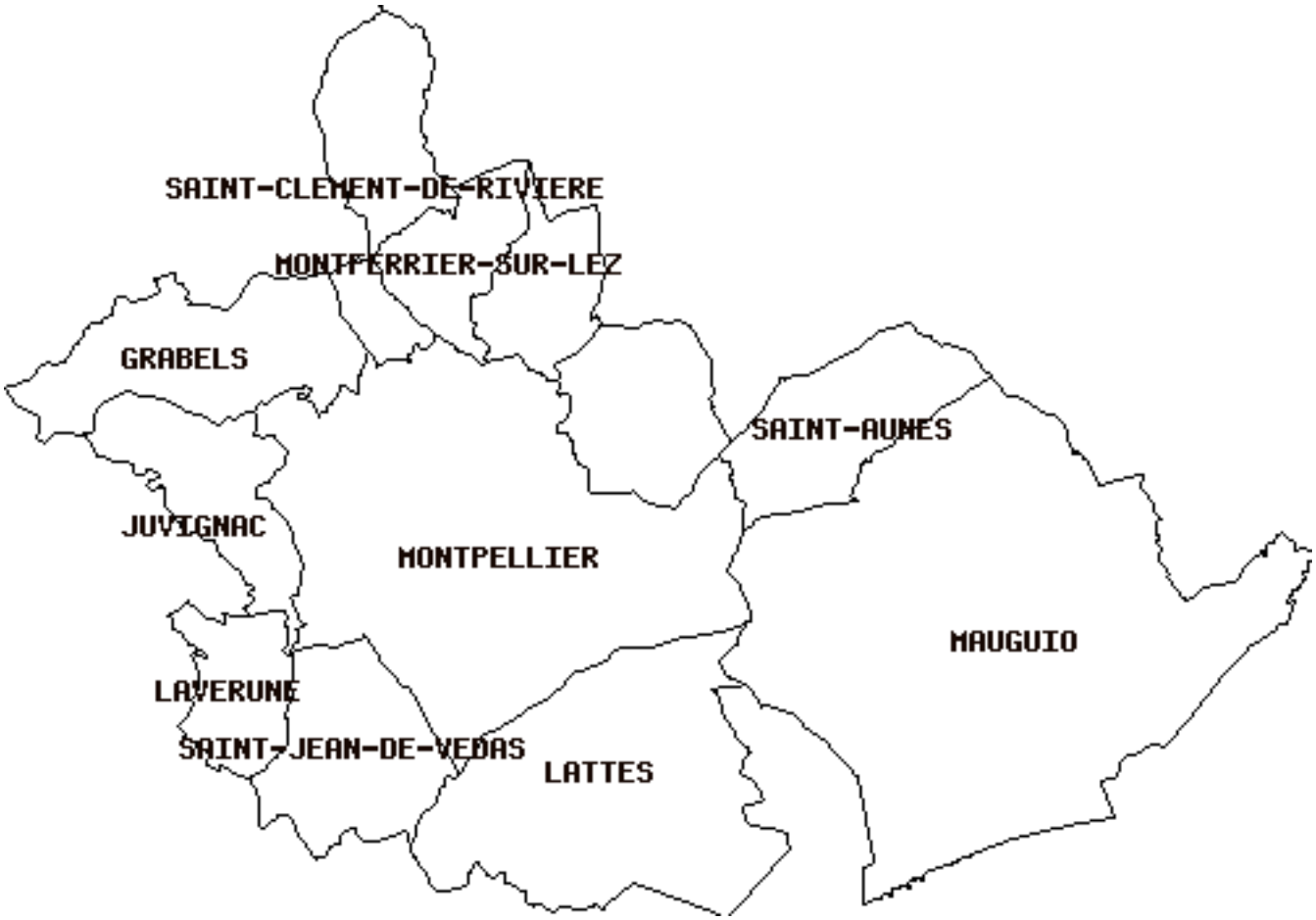

FIG. 2.17 – Les communes avoisinantes de Montpellier

2.6.8 Utilité des index spatiaux - temps demandé pour exécuter la requête

Pour mettre en évidence le temps mis par PostgreSQL pour exécuter la requête de la section précédente avec et sans la condition **a.the_geom && b.the_geom**, le mieux serait donc de reprendre la requête en la combinant aux mot-clés **EXPLAIN ANALYZE**

NOTE
Je n'exposerais pas ici les explications faisant appel au planificateur de tâches et à l'optimisateur de requêtes de PostgreSQL. Le mieux à mon sens est de se référer à la documentation officielle de PostgreSQL.

2.6.8.1 Avec la condition pour &&

La requête suivante

```
EXPLAIN ANALYZE
SELECT b.nom  FROM communes_lr a, communes_lr b
WHERE
     a.nom LIKE 'MONTPELLIER'
AND
     b.nom NOT LIKE 'MONTPELLIER'
AND
    Distance(a.the_geom,b.the_geom)=0
AND
```

```
   a.the_geom && b.the_geom
ORDER BY b.nom
```

nous renvoit comme réponse

```
                                                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------

 Sort  (cost=131.34..131.35 rows=1 width=32) (actual time=932.000..932.000 rows=11 loops=1)
   Sort Key: b.nom
   ->  Nested Loop  (cost=0.00..131.33 rows=1 width=32) (actual time=230.000..921.000 rows ↩
       =11 loops=1)
         Join Filter: (distance("outer".the_geom, "inner".the_geom) = 0::double precision)
         ->  Seq Scan on communes_lr a  (cost=0.00..83.31 rows=8 width=32) (actual time ↩
             =10.000..10.000 rows=1 loops=1)
               Filter: ((nom)::text ~~ 'MONTPELLIER'::text)
         ->  Index Scan using communes_lr_the_geom_gist on communes_lr b  (cost=0.00..5.98  ↩
             rows=1 width=64) (actual time=10.000..20.000 rows=17 loops=1)
               Index Cond: ("outer".the_geom && b.the_geom)
               Filter: ((nom)::text !~~ 'MONTPELLIER'::text)
 Total runtime: 932.000 ms
(10 lignes)
```

soit un temps de 0.9 secondes (cf `Total runtime : 932.000 ms`). On voit bien d'après les résultats que l'index spatiaux est utilisé (cf "`Index Scan using communes_lr_the_geom_gist on communes_lr`", index créé automatiquement lors de l'importation des données par l'option -I de shp2pgsql.)

2.6.8.2 Sans la condition pour &&

La même requête mais sans la contrainte

```
EXPLAIN ANALYZE
SELECT b.nom  FROM communes_lr a, communes_lr b
WHERE
     a.nom LIKE 'MONTPELLIER'
AND
     b.nom NOT LIKE 'MONTPELLIER'
AND
    Distance(a.the_geom,b.the_geom)=0
ORDER BY b.nom
```

renvoit comme résultat

```
                                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------

 Sort  (cost=476.08..476.23 rows=62 width=32) (actual time=152159.000..152159.000 rows=11  ↩
     loops=1)
   Sort Key: b.nom
   ->  Nested Loop  (cost=83.32..474.23 rows=62 width=32) (actual time ↩
       =74577.000..152159.000 rows=11 loops=1)
         Join Filter: (distance("inner".the_geom, "outer".the_geom) = 0::double precision)
         ->  Seq Scan on communes_lr b  (cost=0.00..83.31 rows=1538 width=64) (actual time ↩
             =0.000..50.000 rows=1544 loops=1)
               Filter: ((nom)::text !~~ 'MONTPELLIER'::text)
         ->  Materialize  (cost=83.32..83.40 rows=8 width=32) (actual time=0.006..0.013  ↩
             rows=1 loops=1544)
               ->  Seq Scan on communes_lr a  (cost=0.00..83.31 rows=8 width=32) (actual  ↩
                   time=10.000..20.000 rows=1 loops=1)
                     Filter: ((nom)::text ~~ 'MONTPELLIER'::text)
 Total runtime: 152189.000 ms
```

```
(10 lignes)
```

ce qui s'apparente à un temps d'exécution d'environ 2 minutes 30 (cf. `Total runtime : 152189.000 ms`). Ce qui est énorme ! ! ! Ici, comme attendu il n'est pas fait usage des index spatiaux - par rapport à l'opérateur de chevauchement && -.

Ici par rapport au résultat avec l'opérateur && de la section précédente, la ligne pour la table `'communes_lr b'`

```
->  Seq Scan on communes_lr b  [... ] (actual time=0.000..50.000 rows=1544 loops=1) [...]
```

implique que 1544 enregistrements (cf `rows=1544`) de la table communes_lr ont été testés. Or si on enlève l'enregistrement qui correspond à celui de MONTPELLIER ('condition de la requête `a.nom LIKE 'MONTPELLIER'`), cela fait 1545 enregistrements. Soit le total d'enregistrement contenu dans la table communes_lr. Ce qui suppose donc que tous les enregistrements de la table ont été testés ! Ce qui est justement l'opposé du bénéfice des index en bases de données.

En effet, avec l'opérateur && le coût de la table 'communes_lr b' s'élève à 17 enregistrements qui sont testés !

```
->  Index Scan using communes_lr_the_geom_gist on communes_lr b [...]
(actual time=10.000..20.000 rows=17 loops=1)
```

Résultat qui exploite les index spatiaux !

2.6.9 Créer une table communes_avoisinantes correspondant aux communes avoisinantes de MONTPELLIER, extraite et conforme à la structure de la table communes_lr, exploitable par MapServer.

On pourrait se contenter d'entrée de dire que la réponse directe serait de faire à la volée

```
CREATE TABLE communes_avoisinantes AS
(
   SELECT  b.* FROM communes_lr a, communes_lr b
      WHERE
          a.nom LIKE 'MONTPELLIER'
      AND
         Distance(a.the_geom,b.the_geom)=0
      AND
         a.the_geom && b.the_geom
ORDER BY b.nom
)
```

Le premier souci qui se pose est que l'on n'a pas ici les méta-données concernant la nouvelle table référencées dans la table geometry_columns. On peut s'arrêter là si on n'a pas besoin d'utiliser cette table pour un affichage ultérieure avec MapServer.

Or dans le cas contraire d'un affichage avec MapServer, il est nécessaire que les métadonnées concernant la nouvelle table soient fournies dans la tables geometry_columns. En effet, quand il s'agit de données PostGIS à afficher, MapServer se sert de la fonction `find_srid()` qui elle s'appuie sur la table geometry_columns pour connaître l'identifiant de projection à utiliser.

Le mieux est alors d'avoir recours à quelques requêtes basique.s. On pourra essayer les requêtes successives décrites ici pour répondre à tous ces impératifs :

```
/*
   On commence par effacer la table communes_avoisinantes
   si elle exite déjà.
   Ceci au cas où nous serions amenés à recharger ce fichier.
   On peut aussi utiliser la commande: DROP TABLE communes_avoisinantes
*/
   SELECT drop_table_if_exists('communes_avoisinantes',false);
/*
   On crée une table vide communes_avoisinantes
  pour cela, il suffit de demander de retourner la table communes_lr avec 0 lignes.
  On obtient ainsi la même structure que la table communes_lr
*/
```

```
CREATE TABLE communes_avoisinantes AS
       SELECT * FROM communes_lr LIMIT 0;
/*
  On efface la colonne geometrique 'the_geom' dans la table communes_avoisinantes
  car pour l'instant celle-ci n'est par référencée comme il faut
  dans la table geometry_columns
*/
ALTER TABLE communes_avoisinantes DROP COLUMN the_geom;
/*
  On ajoute la colonne 'the_geom' proprement en utilisant AddGeometryColumn()
  Ici, on joue le jeu que l'on ne connaît de la table
  communes_lr que son nom. On ignore donc le type
  géométrique, son srid...Toutes ces informations sont stockées
  dans la table geometry_columns.
*/
SELECT AddGeometryColumn('communes_avoisinantes'::varchar,
                            foo.f_geometry_column::varchar,
                            foo.srid::integer,
                            foo.type::varchar,
                            foo.coord_dimension::integer
                        )
FROM (
        SELECT  * FROM geometry_columns WHERE
                f_table_name LIKE 'communes_lr'
     ) AS foo;
/*
   On insère maintenant les données attendues dans la table
*/
INSERT INTO communes_avoisinantes
(
   SELECT  b.* FROM communes_lr a, communes_lr b
      WHERE
          a.nom LIKE 'MONTPELLIER'
      AND
         Distance(a.the_geom,b.the_geom)=0
      AND
         a.the_geom && b.the_geom
ORDER BY b.nom
);
```

NOTE
A partir de l'étude réalisée pour la table communes_lr, il est alors aisé de pouvoir visualiser le contenu de cette table dans MapServer.

2.6.10 Requête 1 : Qu'elle est l'intersection entre MONTPELLIER et les communes de LATTES et de JUVIGNAC ?- Intersection()- Que vaut cette géométrie en SVG ? - AsSVG(),

Nous ici pour cela utiliser la table communes_avoisinantes. Toutes ces résultats peuvent être obtenue grâce à la requête suivante :

```
SELECT fooo.nom,
       AsText(Intersection(foo.the_geom,fooo.the_geom)),
       AsSVG(Intersection(foo.the_geom,fooo.the_geom))
FROM (
      SELECT the_geom FROM communes_avoisinantes
           WHERE nom LIKE 'MONTPELLIER'
     ) AS foo
      ,
```

```
    (
     SELECT nom,the_geom FROM communes_avoisinantes
       WHERE nom LIKE 'JUVIGNAC' OR nom LIKE 'LATTES'
    ) AS fooo ;
```

ou sinon la requête suivante possible

```
SELECT b.nom,
       AsText(Intersection(a.the_geom,b.the_geom)),
       AsSVG(Intersection(a.the_geom,b.the_geom))
FROM communes_avoisinantes a,communes_avoisinantes b
WHERE a.nom='MONTPELLIER'
AND (b.nom='LATTES' OR b.nom='JUVIGNAC');
```

Je n'afficherais pas ici les résultats obtenus car ces derniers prennent trop de place en affichage. Je me contenterais de les résumés ainsi

```
   nom    |                              intersection  ↩
                                                     |  ↩
                                          assvg
----------+-------------------------------------------------------------------------------------------

 JUVIGNAC | MULTILINESTRING((719652.990195505 1844379.97900231,[...]718841.967178608  ↩
     1849682.04627155)) | M 719652.9901955052 -1844379.9790023109 [...] 718841.96717860829  ↩
     -1849682.0462715544
 LATTES   | MULTILINESTRING((729577.052318637 1845119.97019685,[...],723311.014479515  ↩
     1841729.00265142))| M 729577.05231863656 -1845119.9701968494[...] 723311.01447951456  ↩
     -1841729.0026514169
(2 lignes)
```

Au niveau de l'affichage on obtient

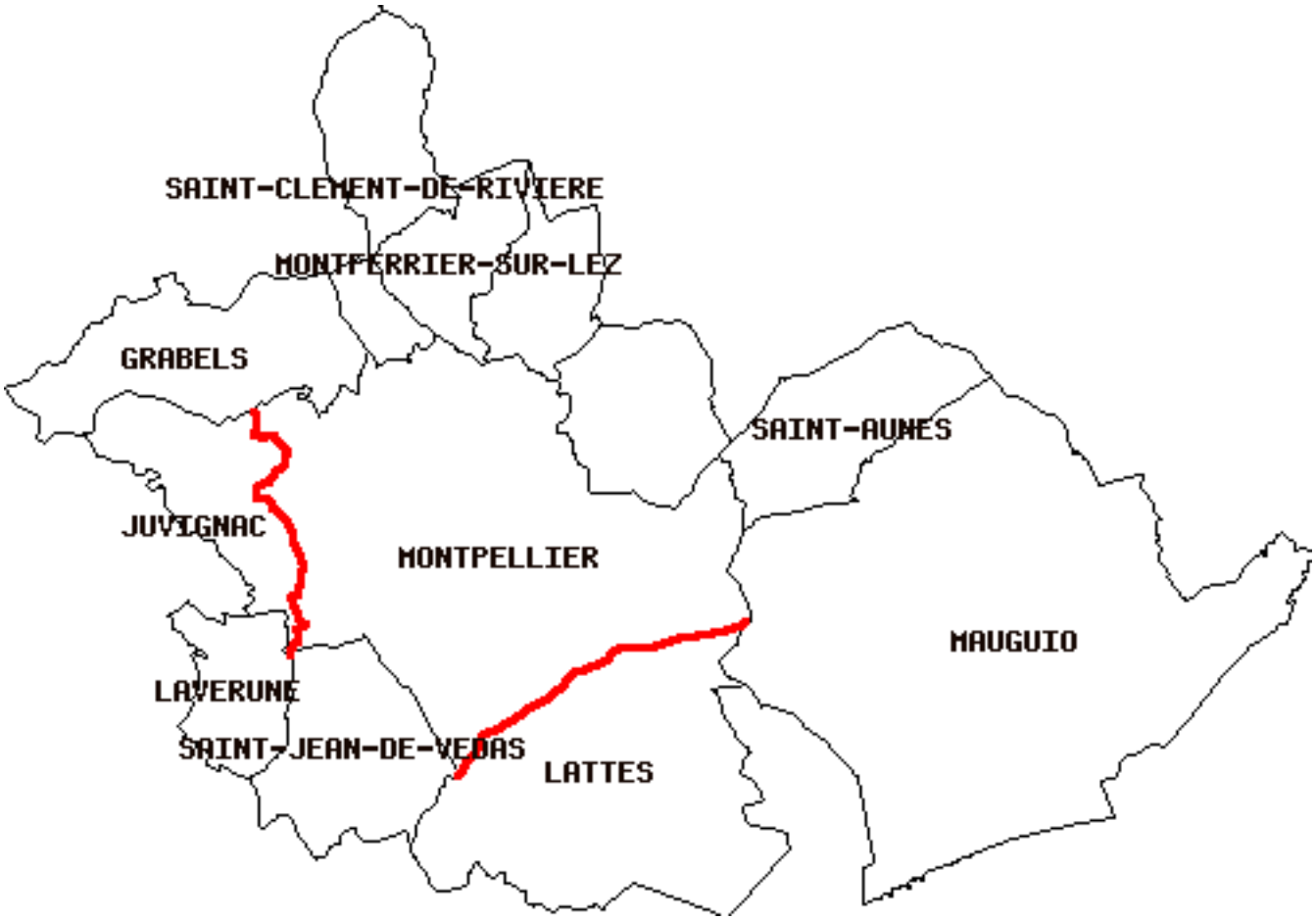

FIG. 2.18 – Intersection entre MONTPELLIER et les comunes de LATTES et de JUVIGNAC

2.6.11 Requête 2 : Qu'elle est la commune ayant la plus petite aire ?

Il suiif par exemple d'utiliser la fonction Min() et Area2d() pour avoir comme requête :

```
SELECT nom FROM communes_avoisinantes
    WHERE Area2d(the_geom) = (SELECT Min(Area2d(the_geom)) FROM communes_avoisinantes)
```

nous renvoyant comme réponse

```
   nom
----------
 LAVERUNE
(1 row)
```

Le visuel sera alors le suivant

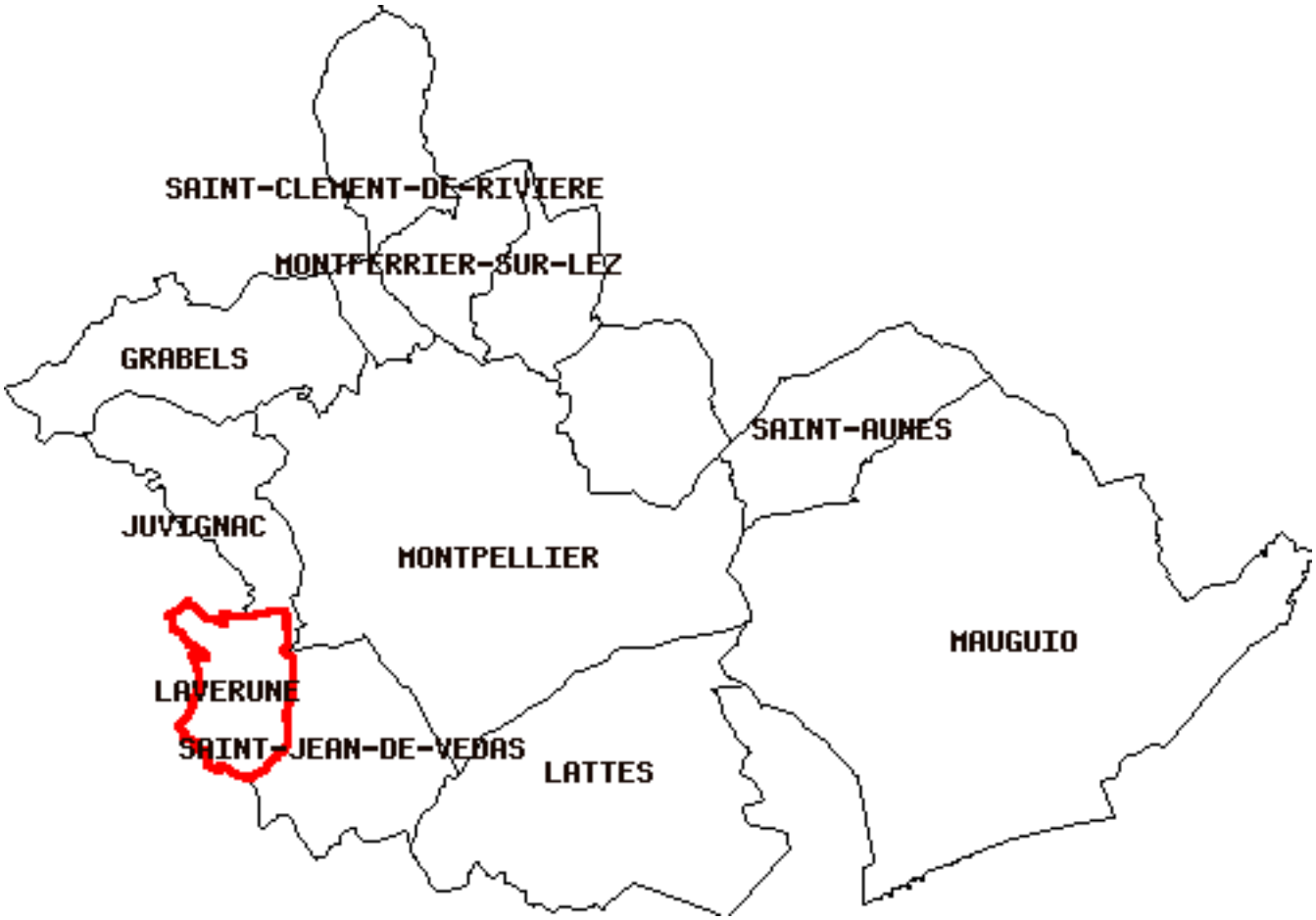

FIG. 2.19 – LATTES : la commune ayant la plus petite aire.

2.6.12 Mapfile générale pour la table communes_avoisinantes et les deux requêtes précédentes

Je fournis ci après la mapfile qui permet d'afficher la table communes_avoisinantes, ainsi que les deux requêtes précédemment poser. Il est à noter surtout la formulation des requêtes - paramètre DATA - pour pouvoir afficher les objets attendues.

```
MAP
  EXTENT 713413.9375 1838659.875 741999.0625 1858572.125
  FONTSET "c:\wamp\www\tutorial\etc\fonts.txt"
  SYMBOLSET "c:\wamp\www\tutorial\etc\symbols.sym"
  IMAGECOLOR 255 255 255
  IMAGETYPE png
  SIZE 500 348.297409929115
#  STATUS ON

  WEB
    IMAGEPATH "c:/wamp/www/tutorial/tmp/"
    IMAGEURL "/tutorial/tmp/"
  END
#============================================================================
# La table communes_avoisinantes
#============================================================================
  LAYER
    NAME "communes_avoisinantes"
    CONNECTION "user=david dbname=madatabase host=localhost"
    CONNECTIONTYPE POSTGIS
```

```
    DATA "the_geom from communes_avoisinantes"
    STATUS DEFAULT
    TYPE POLYGON
    LABELITEM "nom"
    CLASS
      LABEL
        SIZE MEDIUM
        TYPE BITMAP
        BUFFER 0
        COLOR 22 8 3
        FORCE FALSE
        MINDISTANCE -1
        MINFEATURESIZE -1
        OFFSET 0 0
        PARTIALS TRUE
        POSITION CC
      END
      STYLE
          COLOR 255 255 255
          OUTLINECOLOR 0 0 0
      END
    END
  END
#=============================================================================
# Requête 1: Intersection entre Montpellier et les communes de LATTES et JUVIGNAC
#=============================================================================
  LAYER
    NAME "requete_1"
    CONNECTION "user=david dbname=madatabase host=localhost"
    CONNECTIONTYPE POSTGIS
    DATA "intersection from
    (SELECT
       intersection(foo.the_geom,fooo.the_geom),fooo.nom,fooo.gid
FROM (
      SELECT * FROM communes_avoisinantes
           WHERE nom LIKE 'MONTPELLIER'
     ) AS foo
      ,
     (
      SELECT * FROM communes_avoisinantes
        WHERE nom LIKE 'JUVIGNAC' OR nom LIKE 'LATTES'
     ) AS fooo
)AS uo USING UNIQUE gid USING SRID=27582"
    STATUS OFF
    TYPE LINE
    LABELITEM "nom"
    CLASS
      STYLE
        OUTLINECOLOR 255 0 0
        SIZE 3
        SYMBOL "circle"
      END
    END
  END
#=============================================================================
# Requête 2: La plus petite commune autour de Montpellier
#=============================================================================
  LAYER
    NAME "requete_2"
    CONNECTION "user=david dbname=madatabase host=localhost"
    CONNECTIONTYPE POSTGIS
    DATA "the_geom from
```

```
    (SELECT the_geom,nom,gid FROM communes_avoisinantes
    WHERE Area2d(the_geom) = (SELECT Min(Area2d(the_geom)) FROM
    communes_avoisinantes))AS uo USING UNIQUE gid USING SRID=27582"
    STATUS DEFAULT
    TYPE LINE
    LABELITEM "nom"
    CLASS
      STYLE
        OUTLINECOLOR 255 0 0
        SIZE 3
        SYMBOL "circle"
      END
    END
  END

END
```

2.6.13 Exercice : Obtenir une table departements_lr qui contient les contours départmentaux du Languedoc-Roussillon à partir de la table communes_lr

La région du Languedoc-Roussillon est composée de 5 départements dont voici la liste

Aude 11 ;

Hérault 34 ;

Gard 30 ;

Lozère 48 ;

Pyrénées Orientales 66.

Normalement lorsqu'on dispose des limites communales d'une région on doit aussi pouvoir disposer des limites départementales de cette dernière. Nous partirons ici du principe que nous ne disposons que de la table communes_lr. C'est à partir de cette table que nous allons créer nos limites départementales. Nous allons mettre les contours départementaux dans une table **departements_lr** ayant la structure suivante

```
CREATE TABLE departements_lr(id serial,nom text,numero integer);
```

où nom désignera le nom du département et numero son numéro. Comme pour l'instant, je ne sais pas quel est la la nature des objets géométrique, je fais juste

```
SELECT AddGeometryColumn('departements_lr_tmp','the_geom',27582,'GEOMETRY',2);
```

Je remplis maintenant la table pour tous les départements

```
INSERT INTO departements_lr
(
select 1 as id,'Herault'::text as nom,34 as numero,geomunion(the_geom) as the_geom from
communes_lr where insee like '34%'
);
INSERT INTO departements_lr
(
select 2 as id,'Gard'::text as nom,30 as numero,geomunion(the_geom) as the_geom from
 communes_lr where insee like '30%'
);
INSERT INTO departements_lr
(
select 3 as id,'Lozère'::text as nom,48 as numero,geomunion(the_geom) as the_geom
from communes_lr where insee like '48%'
);
INSERT INTO departements_lr
(
```

```
select 4 as id,'Aude'::text as nom,11 as numero,geomunion(the_geom) as the_geom from
communes_lr where insee like '11%'
);
INSERT INTO departements_lr
(
select 5 as id,'Pyrénées Orientales'::text as nom,66 as numero,geomunion(the_geom)
as the_geom from communes_lr where insee like '66%'
);
```

Je ne cache pas qu'ici le peuplement de la table m'a pris 7 à 8 minutes quand même ! Compréhensible dans le sens où faire/calculer la réunion géométrique de plusieurs objets géométriques demande du temps.

Pour savoir quel est le type de données géométriques adéquates, il faut utiliser la commande

```
select nom,geometrytype(the_geom) from departements_lr
```

qui me renvoit

```
         nom          | geometrytype
----------------------+--------------
 Herault              | MULTIPOLYGON
 Gard                 | POLYGON
 Lozère               | POLYGON
 Aude                 | POLYGON
 Pyrénées Orientales | POLYGON
(5 lignes)
```

Je réordonne les tables geometry_columns en mettant le type de departements_lr à MULTIPOLYGON et je convertis tous les departements en MULTIPOLYGON grâce à la fonction `Multi()` de PostGIS.

```
update geometry_columns set type='MULTIPOLYGON' where f_table_name='departements_lr';
update  departements_lr set the_geom=(multi(the_geom::text));
```

il faut ensuite créer l'index spatial sur la table departements_lr et faire un vacuum analyze sur la base.

```
create index departements_lr_the_geom_gist on departements_lr using gist(the_geom  ↩
    gist_geometry_ops);
vacuum analyze;
```

Au niveau de la mapfile, il faudrait ajouter le layer suivant

```
  #=========================================================
  # Couche des départements
  #=========================================================
  LAYER
    NAME "departements_lr"
    CONNECTION "user=david dbname=madatabase host=localhost"
    CONNECTIONTYPE POSTGIS
    DATA "the_geom from departements_lr"
    STATUS DEFAULT
    TYPE POLYGON
    LABELITEM "nom"
    CLASS
       LABEL
        SIZE MEDIUM
        TYPE BITMAP
        BUFFER 0
        COLOR 22 8 3
        FORCE FALSE
        MINDISTANCE -1
        MINFEATURESIZE -1
        OFFSET 0 0
```

```
        PARTIALS TRUE
        POSITION CC
      END
      STYLE
        OUTLINECOLOR 255 0 0
        SIZE 3
        SYMBOL "circle"
      END
    END
  END
```

AI niveau de l’afffichage, on obtient alors

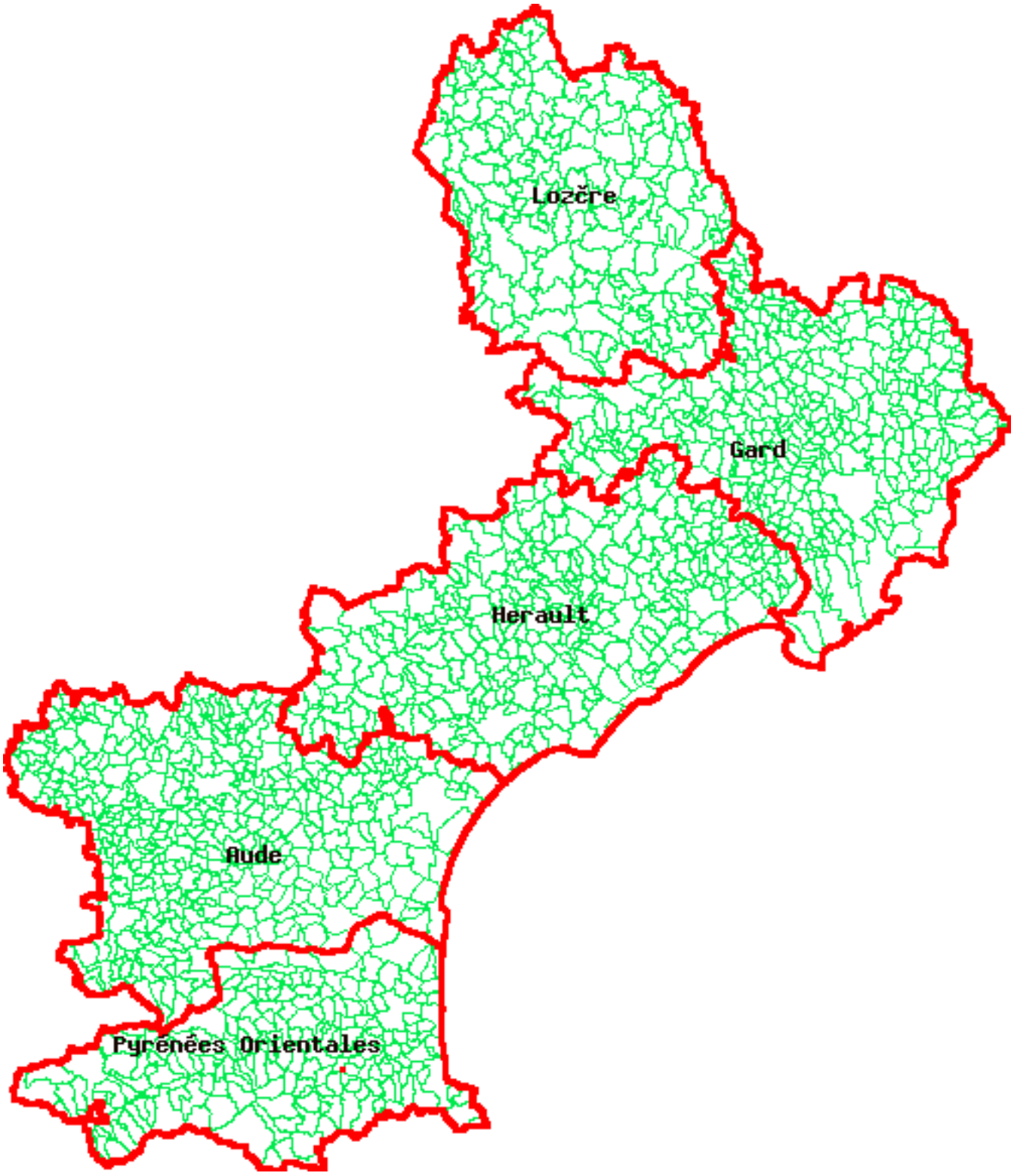

FIG. 2.20 – Affichage des départements du Languedoc-Roussillon

2.6.14 Exercice : Trouver les communes du Gard et de l'Aude qui sont limitrophes à l'Hérault et les afficher grâce à MapServer.

Comme je sais ici que les communes des départements du Gard et de l'Aude commencent respectivement par 30 et 11, je vais pour cela utiliser la fonction Touches de PostGIS.

Au niveau de la mapfile, je ferais simplement

```
#============================================================
# requete
#============================================================
LAYER
  NAME "requete"
  CONNECTION "user=david dbname=madatabase host=localhost"
  CONNECTIONTYPE POSTGIS
  DATA "the_geom from (
    select  c.*  from departements_lr d,communes_lr c
   where  touches(d.the_geom,c.the_geom)
   and d.the_geom && c.the_geom
   and d.nom='Herault'
  and (c.insee like '11%' or c.insee like '30%')
  ) as foo USING UNIQUE gid USING SRID=27582"
  STATUS DEFAULT
  TYPE POLYGON
  CLASS
     LABEL
      SIZE MEDIUM
      TYPE BITMAP
      BUFFER 0
      COLOR 22 8 3
      FORCE FALSE
      MINDISTANCE -1
      MINFEATURESIZE -1
      OFFSET 0 0
      PARTIALS TRUE
      POSITION CC
    END
    STYLE
      OUTLINECOLOR 0 0 255
      SIZE 2
      SYMBOL "circle"
    END
  END
 END
```

Au niveau de l'affichage avec MapServer, j'obtiens

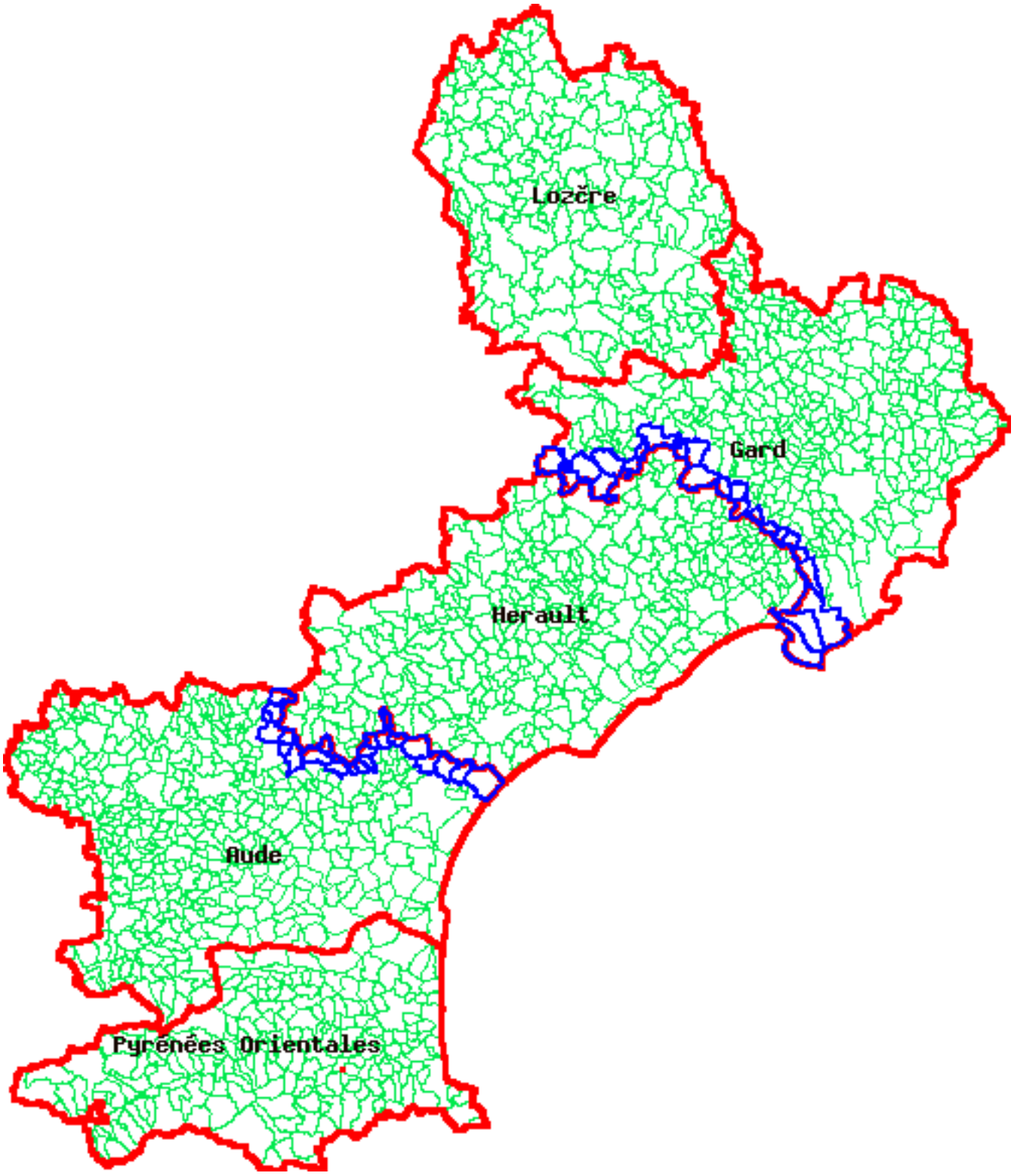

FIG. 2.21 – MapServer : Communes du Gard (30) et de l'Aude (11) limitrophes à l'Hérault (34)

Chapitre 3

Bibliographie

3.1 [1]

[PostGIS Manual] Paul RAMSEY et Sandro Santilli, *PostGIS Manual (http ://postgis.refractions.net)*, Copyright © 2006 PostGIS Team.

3.2 [2]

[Documentation PostgreSQL 8.1.1] "Documentation PostgreSQL 8.1.1 / Traduction de Guillaume LELARGE http ://traduc.postgresqlfr.org/pgsql-8.1.1-fr", Copyright © 1996-2006 The PostgreSQL Global Develop-met Group.