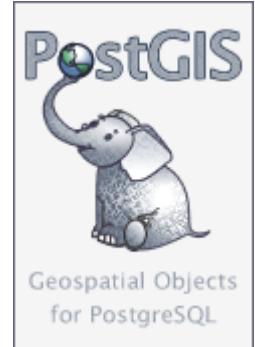




CENTRE D'ECOLOGIE  
FONCTIONNELLE  
& EVOLUTIVE



# Formation PostGIS 1.5

Cyril Bernard ([cyril.bernard@cefe.cnrs.fr](mailto:cyril.bernard@cefe.cnrs.fr))

Centre d'Ecologie Fonctionnelle et Evolutive - UMR 5175  
Montpellier, 27 juin 2012

# Ce document est diffusé sous licence « licence ouverte (open licence) »



LICENCE OUVERTE  
OPEN LICENCE

**Vous pouvez réutiliser le présent document, dans le monde entier et pour une durée illimitée, dans les libertés et les conditions exprimées par la licence.**

**Vous êtes libre de réutiliser ce document, c'est-à-dire :**

- Le reproduire, le copier, le publier et le transmettre ;
- Le diffuser et le redistribuer ;
- L'adapter, le modifier, en extraire des passages et le transformer notamment pour créer des informations dérivées ;
- L'exploiter à titre commercial, par exemple en le combinant avec d'autres créations, ou en l'incluant dans votre propre produit ou application.

**Sous réserve de :**

- Mentionner la paternité de ce document : sa source (*a minima le nom de ses auteurs*) et la date de sa dernière mise à jour.

Les auteurs ne peuvent garantir l'absence de défauts ou d'irrégularités éventuellement contenues dans ce document. Ils ne garantissent pas la fourniture continue de mises à jour. Ils ne peuvent être tenus pour responsables de toute perte, préjudice ou dommage de quelque sorte causé à des tiers du fait de la réutilisation.

Vous êtes le seul responsable de la réutilisation que vous ferez de ce document. La réutilisation ne doit pas induire en erreur des tiers quant à son contenu, sa source et sa date de mise à jour.

**Le texte complet de la licence ouverte (open licence) est disponible sur le site d'ETALAB à l'adresse suivante :**

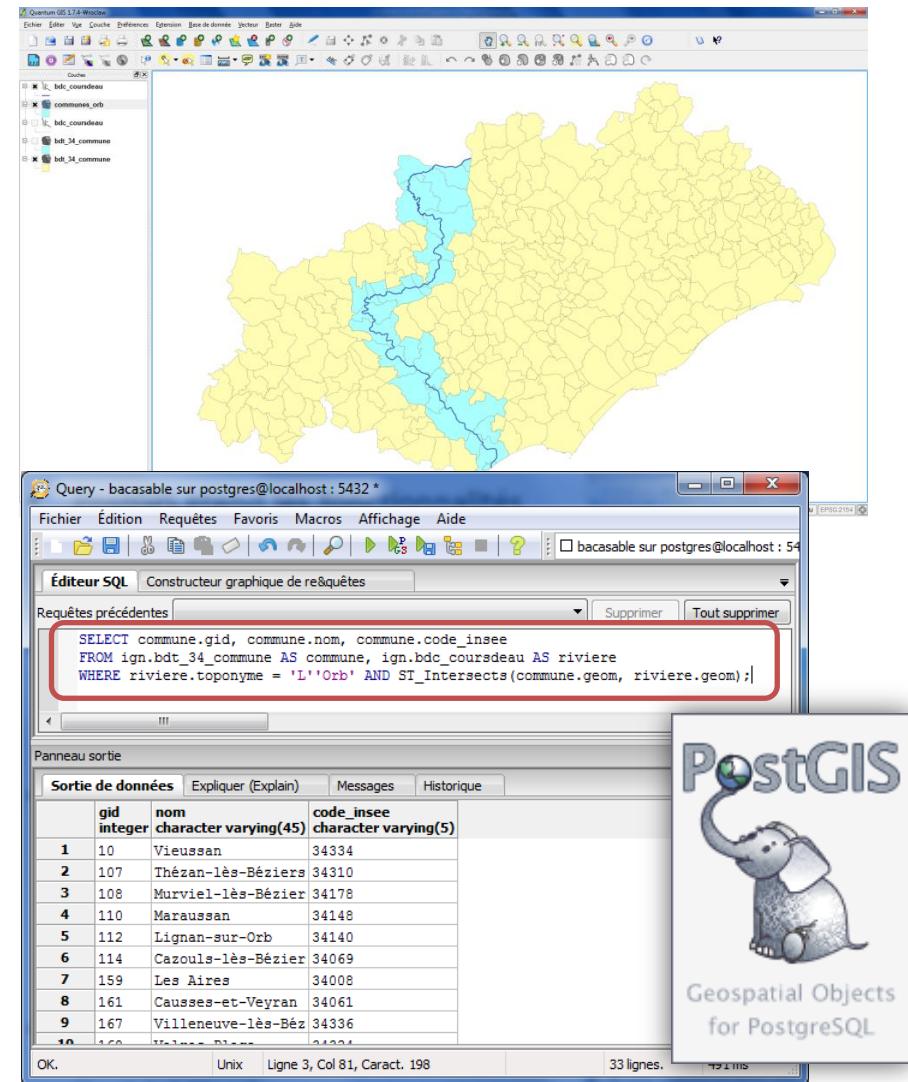
<http://www.etalab.gouv.fr/pages/licence-ouverte-open-licence-5899923.html>

Formation PostGIS 1.5

# **PRÉSENTATION DE POSTGIS**

# Qu'est ce que PostGIS ?

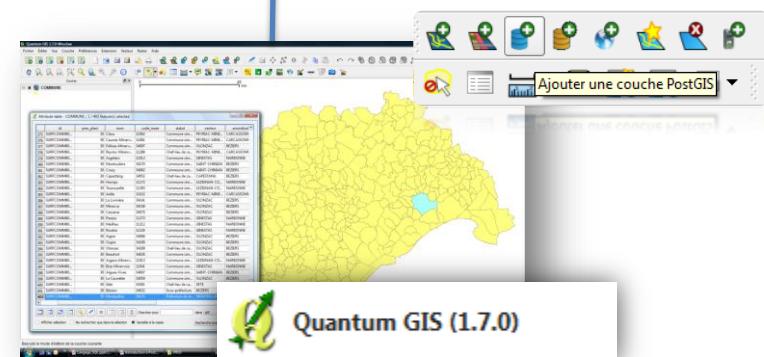
- PostGIS étend les fonctionnalités de PostgreSQL : c'est le module spatial du SGBD
- Permet de stocker des données géographiques dans une base de données PostgreSQL ...
- Permet de faire des requêtes SQL basées sur les relations spatiales
  - Ex : quelles sont les communes traversées par la rivière Orb ? Quelle est la longueur de rivière incluse dans chaque commune ?
- Comme PostgreSQL, PostGIS est distribué sous licence GPL (logiciel libre)



# Qu'est que *n'est pas* PostGIS ?

- PostGIS n'est pas un logiciel de présentation des données : pas d'interface, pas de création de carte
  - Quantum GIS permet de visualiser les données d'une base PostGIS
- PostGIS n'est pas un logiciel de saisie de données
  - Utilisez un SIG Desktop comme Quantum GIS ou une interface web pour la saisie de données

SERVEUR PostgreSQL / PostGIS  
Stockage des données,  
exécution des requêtes

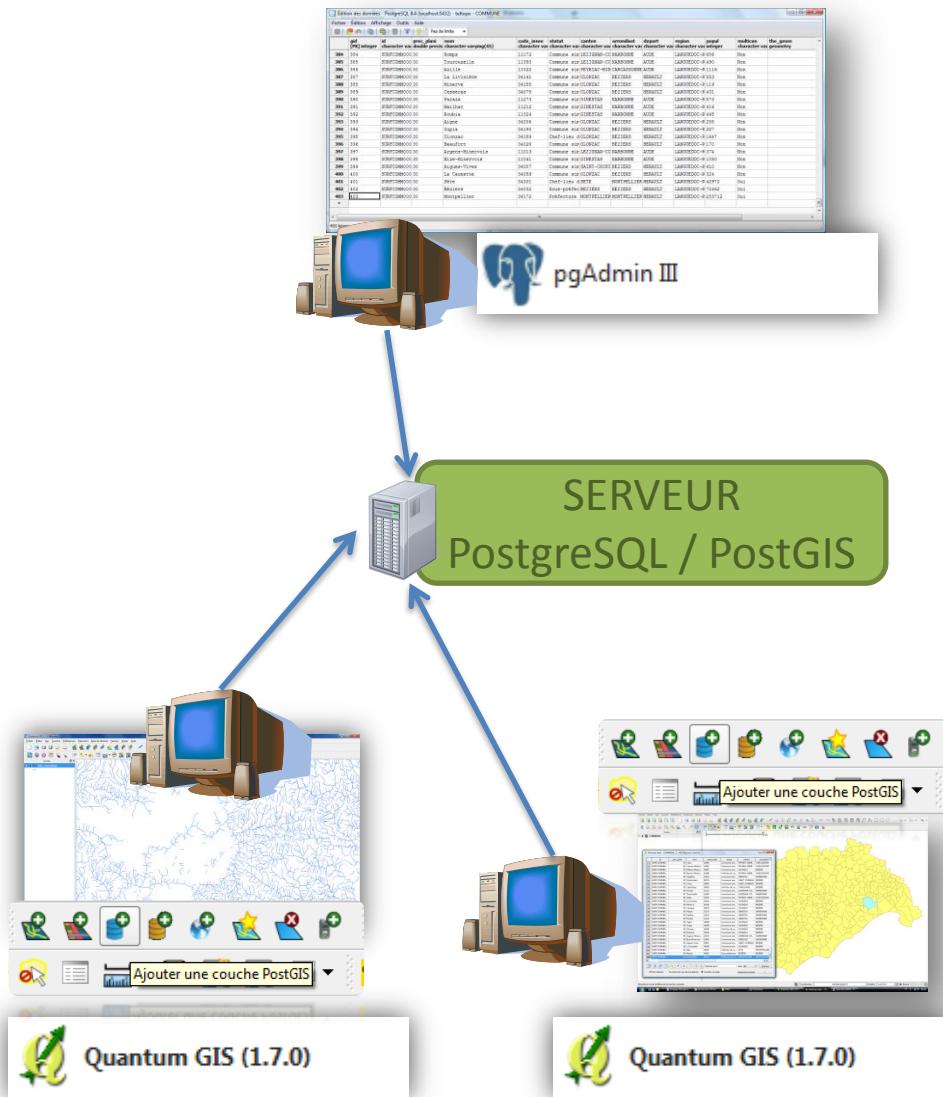


Ordinateur Lambda  
Saisie, visualisation des données,  
lancement des requêtes



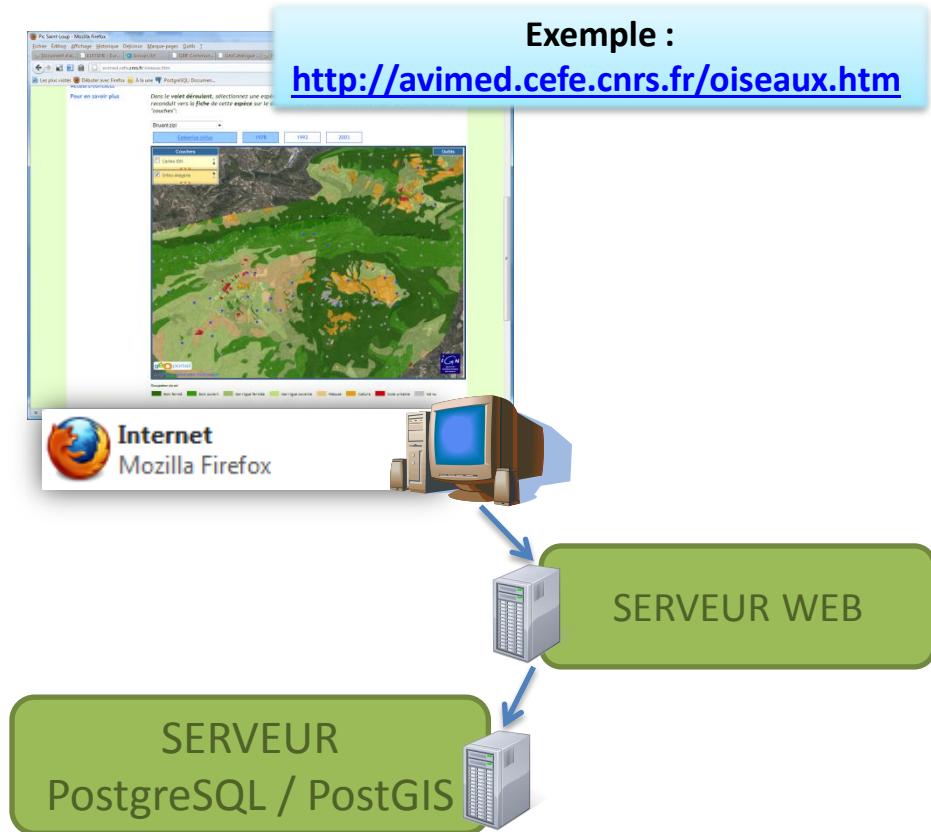
# Cas d'utilisation 1 : PostGIS pour administrer et partager des données SIG

- Plusieurs personnes ont besoin d'accéder / mettre à jour conjointement une base de données commune
  - PostGIS permet de stocker des données SIG dans une BD PostgreSQL et de définir les **droits d'accès** aux données (lecture seule, mise à jour, création, destruction)
  - Quantum GIS 1.7** est conçu pour fonctionner avec PostGIS et s'intègre très bien dans le système d'information pour l'interrogation et la visualisation des données
  - Il est possible d'extraire des données de PostGIS et de les convertir en **shapefile** pour les exploiter dans d'autres SIG



# Cas d'utilisation 2 : PostGIS pour diffuser / collecter des données via Internet

- PostGIS est particulièrement utilisé dans le domaine du webmapping (cartographie interactive sur internet)
  - La diffusion de données se fait généralement via un WMS (Web Map Service) : extraction de données dans une BD PostGIS et transformation dans un format d'image affichable dans un navigateur web (JPG, GIF, PNG)

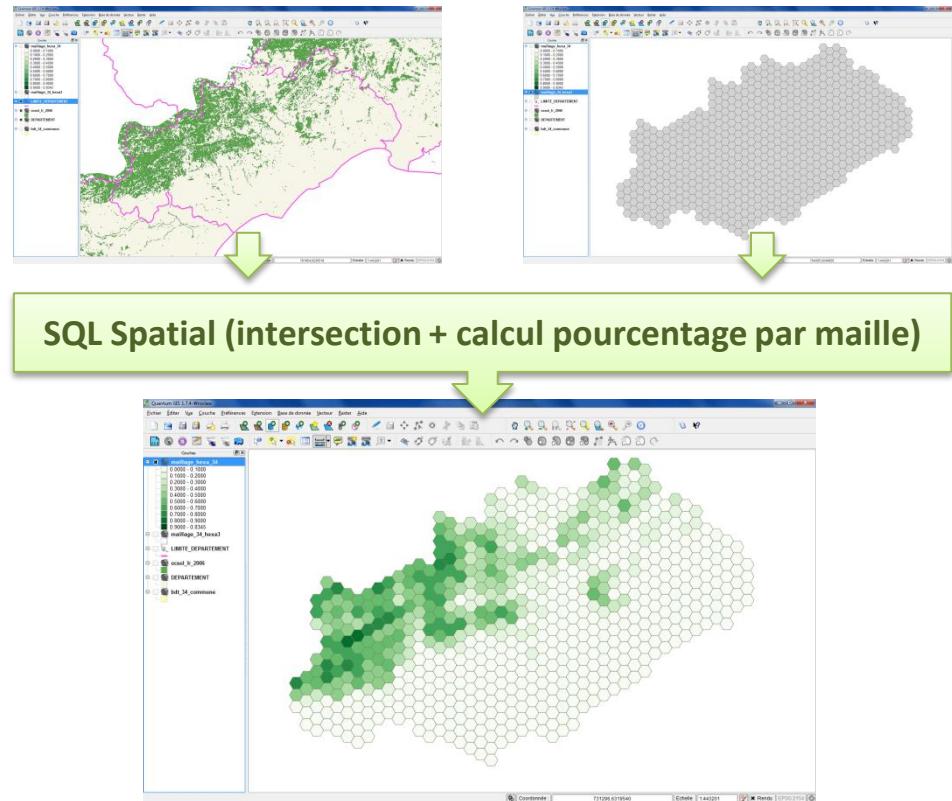


Note : La mise en place de plateformes collaboratives peut s'appuyer sur PostGIS pour le stockage de données cartographiques collectées via internet. La connaissance du langage **Javascript** est cependant nécessaire pour créer des interfaces de saisie cartographique intégrées dans un site web.

# Cas d'utilisation 3 : PostGIS pour les requêtes spatiales, les jointures spatiales et les géotraitements

- Géotraitements sous PostGIS : création de scripts SQL Spatial pour traiter des données SIG (intersection, calcul de surface, relations de distance, etc.)

- Traçabilité : garder une trace des traitements effectués
- Réutilisation des scripts pour d'autres données
- Souplesse : possibilité de scinder un gros jeu de données en plusieurs parties pour le traitement



Note : les fonctions SQL Spatial disponibles dans PostGIS sont issues d'un standard défini par l'OGC (Open Geospatial Consortium) et sont également implémentées dans Oracle Spatial, ArcGIS Server ...

# SIG et SQL ...

## Normes et sigles

- **OGC** = Open Geospatial Consortium. Organismes et entreprises œuvrant dans le domaine de l'information géographique.
- **OpenGIS** = standards définis par l'OGC pour l'interopérabilité dans le domaine de la géomatique (GML, WMS, etc.)
- **Simple Feature for SQL (OGC-SFS)** = un des standards OpenGIS.
- PostGIS implémente le standard OpenGIS « Simple Feature for SQL » ... et l'étend
  - The OpenGIS "Simple Features Specification for SQL" defines standard **GIS object types**, the **functions** required to manipulate them, and a set of **meta-data tables** (**manuel PostGIS, 4.3**)



Le standard SFS dans les géodatabases ESRI :

[http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=An\\_overview\\_of\\_working\\_with\\_ST\\_Geometry\\_storage\\_using\\_SQL](http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=An_overview_of_working_with_ST_Geometry_storage_using_SQL)

Le standard SFS dans Spatialite :

<http://www.gaia-gis.it/spatialite-3.0.0-BETA/spatialite-cookbook-fr/html/wkt-wkb.html>

Formation PostGIS 1.5

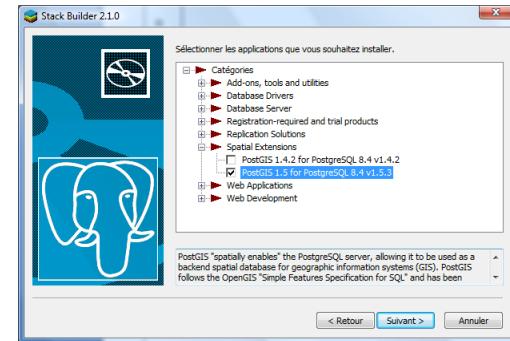
# **INSTALLATION DE POSTGIS CREATION D'UNE BDD SPATIALE**

# Installation de la partie serveur + client sur Windows : PostgreSQL 9.1 + PostGIS 1.5

- 1<sup>ère</sup> étape : installation de **PostgreSQL 9.1 32 bits**

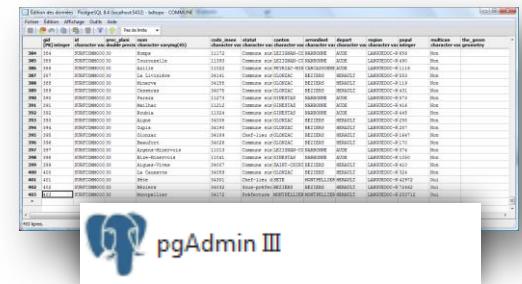
<http://www.enterprisedb.com/products-services-training/pgdownload>

- Installe également pgAdmin III
- bien noter le mot de passe de *postgres* !



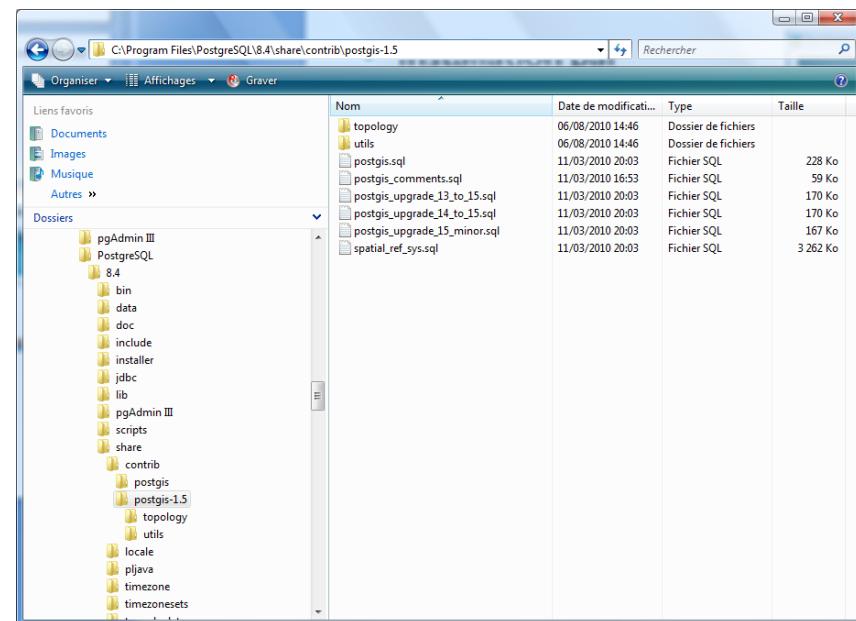
- 2<sup>nde</sup> étape : installation de **PostGIS 1.5** avec **StackBuilder** (assistant de post-installation)

- Et sur Linux Ubuntu : par apt-get install postgis
- Et sur Mac : voir le site <http://www.kyngchaos.com>



# Déroulement de l'installation avec StackBuilder

- De nouveaux fichiers sont installés dans C:\Program Files (x86)\PostgreSQL\9.x
  - Installation des dépendances dans bin : GEOS 3.3, PROJ 4.6
  - Installation de l'utilitaire **shp2pgsql** dans bin
  - Création des scripts **spatial\_ref\_sys.sql** et **postgis.sql** dans share\contrib\postgis-1.5
- Création d'une base **template-postgis** (modèle utilisable pour créer des bases PostGIS vide)
- Création (optionnelle) d'une base exemple vide



# Installation de la partie client seule

- **Quantum GIS 1.7** peut suffire pour une utilisation ‘géomatique simple’ (chargement de données avec SPIT, consultation et requêtes)

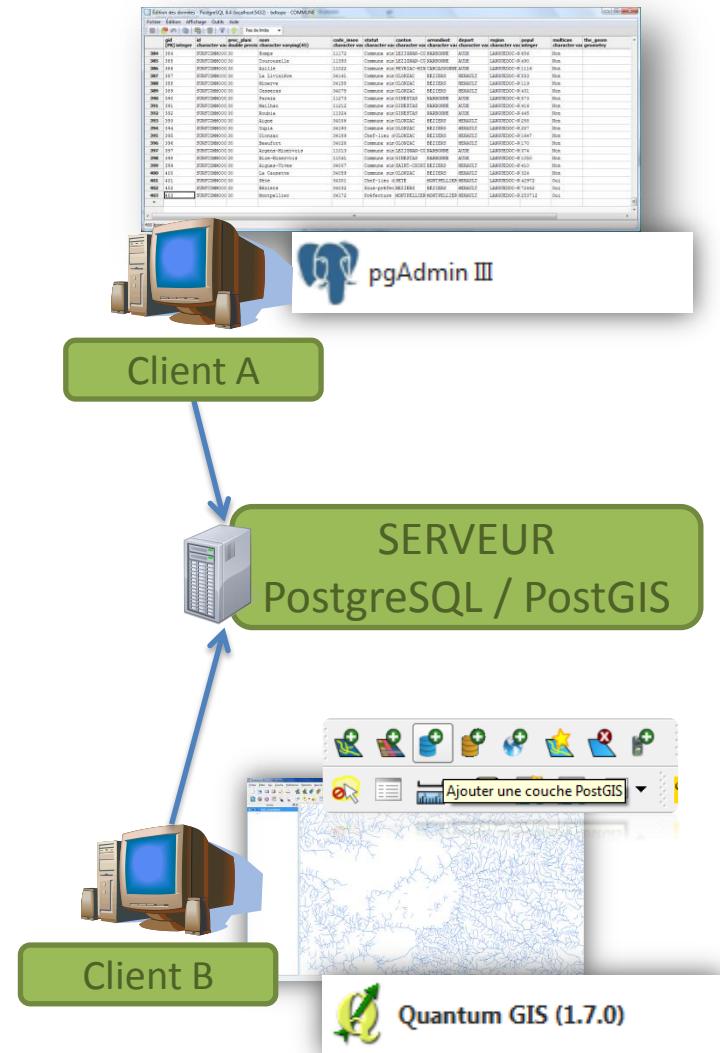
<http://hub.qgis.org/projects/quantum-gis/wiki/DownloadFr> (Windows, installateur indépendant)

- L'installation de **pgAdmin III** est recommandée pour administrer la base (création des objets, gestion des utilisateurs, sauvegarde) et pour exécuter des scripts

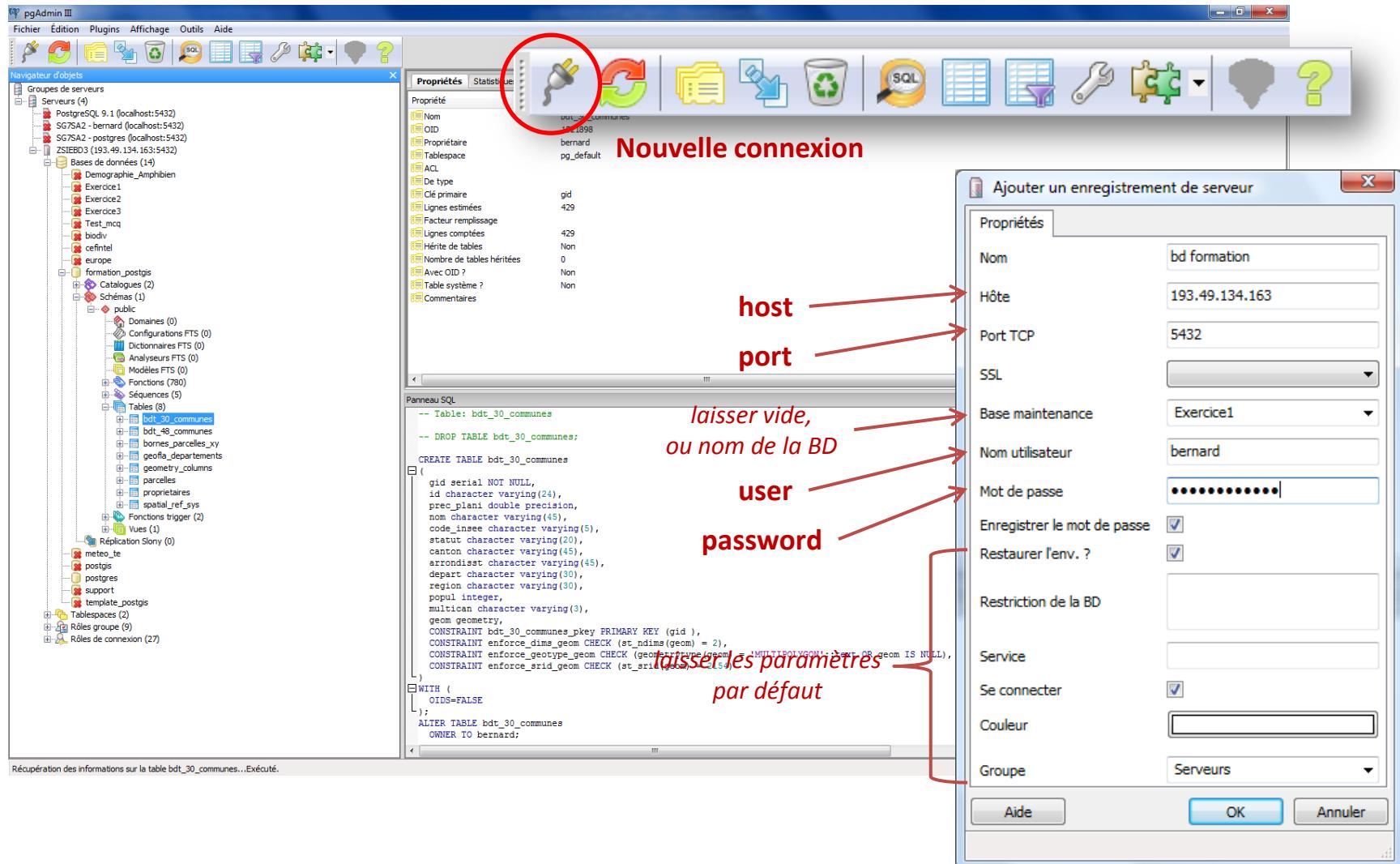
<http://www.postgresql.org/ftp/pgadmin3/release/> (v 1.14.3 / win32)

- L'installer de **shp2pgsql** peut aussi être utile pour importer des shapefiles

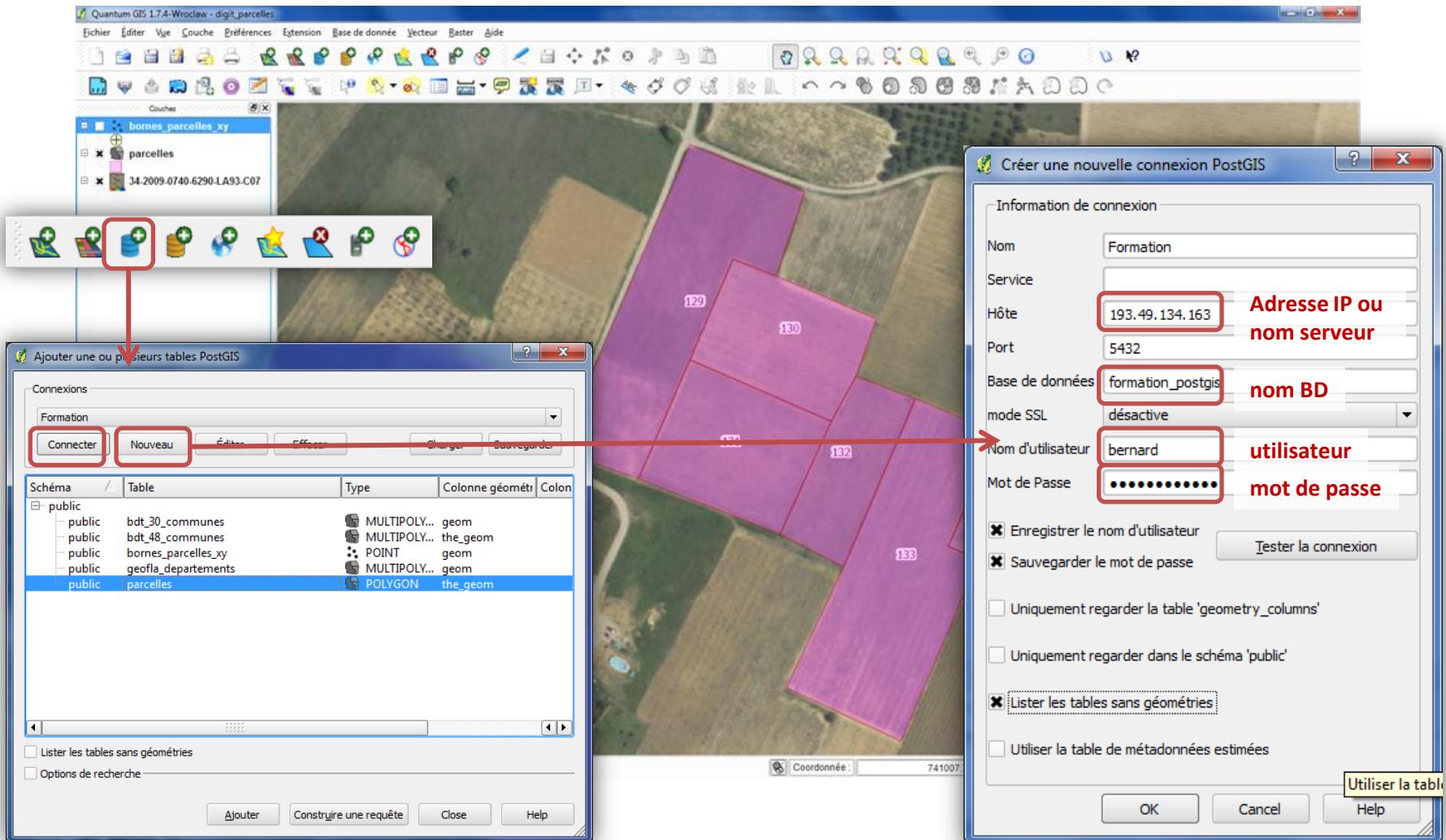
<http://postgis.refractions.net/download/windows/> (PostGIS 2.0.0 Shp2PgSQL Graphical Loader/dumper)



# Connexion à une base de données PostGIS avec pgAdmin III

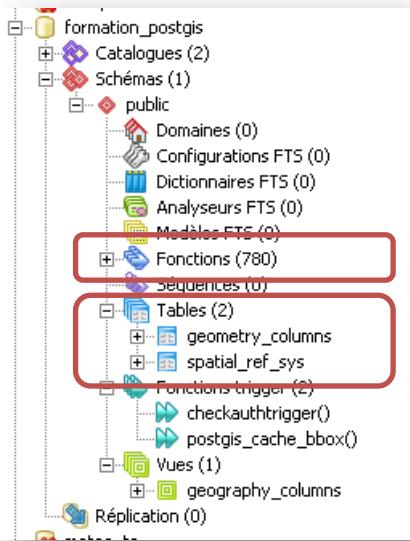


# Connexion à une base de données PostGIS avec Quantum GIS 1.7



# Anatomie d'un BD PostGIS : différence entre BD ‘classique’ et BD ‘spatiale’

- Présence de la table de métadonnées *geometry\_columns*
  - répertorie les colonnes géométriques de la base
  - Crée par postgis.sql
- Présence de la table de métadonnées *spatial\_ref\_sys*
  - répertorie les SRID (définition des systèmes de coordonnées, format PROJ4)
  - Crée par spatial\_ref\_sys.sql
- Définition des fonctions PostGIS
  - 780 à 945 fonctions selon la version ...
  - Crées par postgis.sql



## Exemple de fonctions PostGIS

```
st_geometryfromtext(text, integer)
st_geometryn(geometry, integer)
st_geometrytype(geometry)
st_geomfromewkb(bytea)
st_geomfromewkt(text)
st_geomfromgml(text)
st_geomfromkml(text)
st_geomfromtext(text, integer)
st_geomfromtext(text)
st_geomfromwkb(bytea)
st_geomfromwkb(bytea, integer)
st_gmltosql(text)
st_hasarc(geometry)
st_hausdorffdistance(geometry, geometry)
st_hausdorffdistance(geometry, geometry, double)
st_height(chip)
st_interiorringn(geometry, integer)
st_intersection(geometry, geometry)
st_intersection(geography, geography)
st_intersection(text, text)
st_intersects(geography, geography)
st_intersects(text, text)
st_intersects(geometry, geometry)
st_isclosed(geometry)
st_isempty(geometry)
```

# Création d'une BD Spatiale vide

## Méthode 1 : à partir des scripts postgis.sql et spatial\_ref\_sys.sql

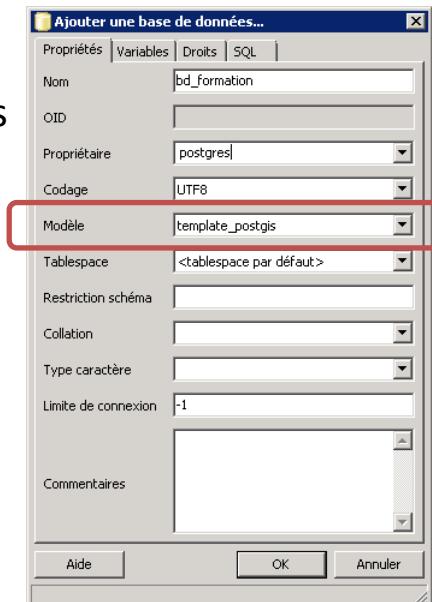
- Exemple depuis l'invite de commande DOS :
  - `createdb options nom_bd`
  - `createlang options plpgsql nom_bd`
  - `psql options -d nom_bd -f chemindacces\postgis.sql`
  - `psql options -d nom_bd -f chemindacces\spatial_ref_sys.sql`
- options de connexion :
  - `-h host -U utilisateur -p port`

```
C:\>cd "C:\Program Files <x86>\PostgreSQL\9.0\bin"
C:\Program Files <x86>\PostgreSQL\9.0\bin>createdb -U postgres formation_postgis
C:\Program Files <x86>\PostgreSQL\9.0\bin>createlang -U postgres plpgsql formation_postgis
createlang : le langage « plpgsql » est déjà installé sur la base de données « formation_postgis »

C:\Program Files <x86>\PostgreSQL\9.0\bin>psql -U postgres -d formation_postgis -f "C:\Program Files <x86>\PostgreSQL\9.0\share\contrib\postgis-1.5\postgis.sql"
```

## Méthode 2 : à partir du modèle template\_postgis

- Exemple depuis une session SQL  
`CREATE DATABASE nom_bd WITH OWNER = postgres  
ENCODING = 'UTF8' TEMPLATE = template_postgis`
- Exemple depuis pgAdmin (clic-droit sur Bases de données + Ajouter une base)

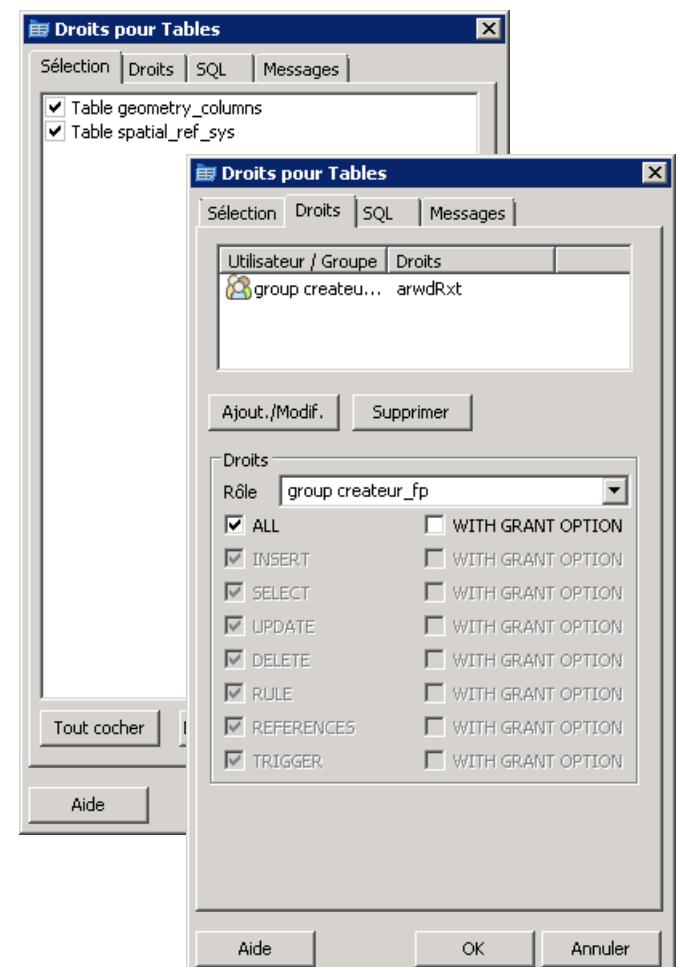
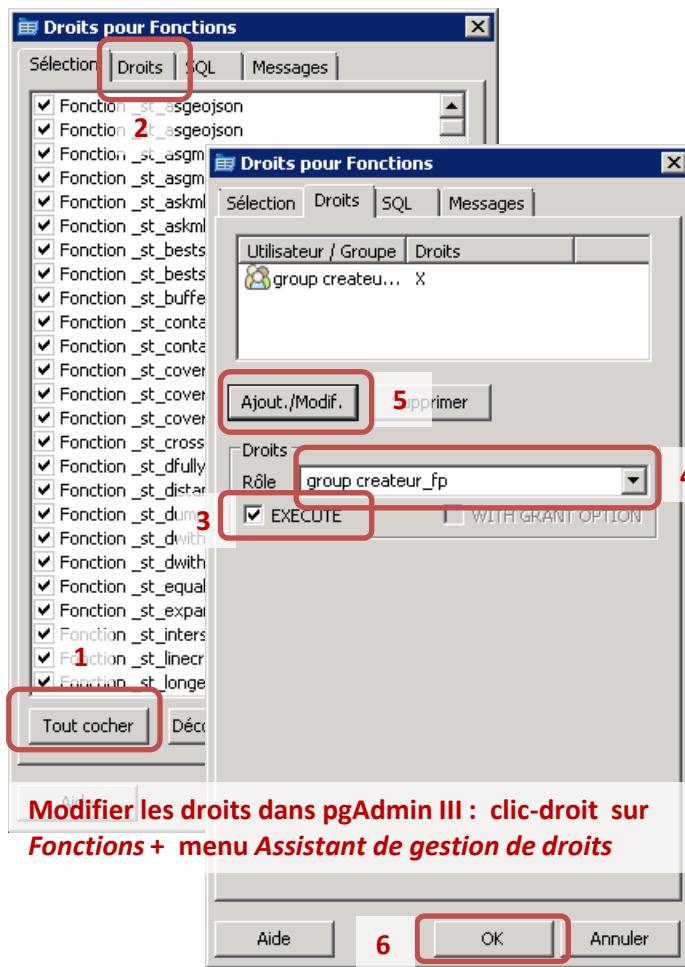


# Questions de droits

Pour consulter les données dans QGIS, un utilisateur lambda doit avoir le droit :

- 1) d'exécuter (EXECUTE) les fonctions PostGIS et
- 2) de lire (SELECT) les tables *geometry\_columns* et *spatial\_ref\_sys*

Pour créer / détruire des colonnes géométrie, il doit avoir tous les droits (ALL) sur ces 2 tables

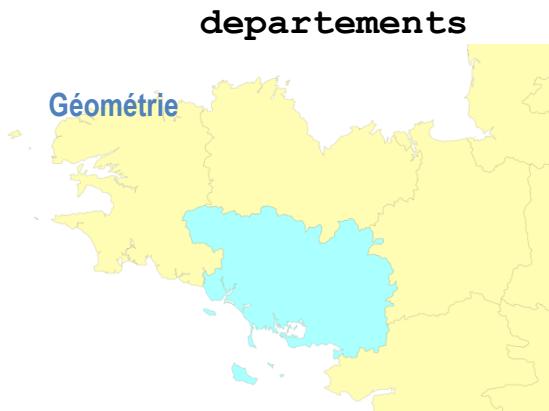


Formation PostGIS 1.5

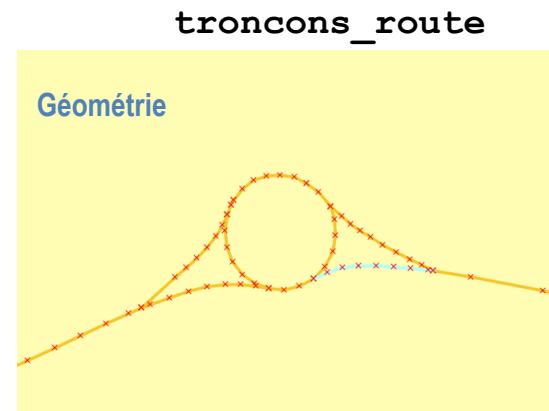
# **CRÉATION ET IMPORT DE DONNÉES SPATIALES**

# Rappel ... qu'est que l'information géographique ?

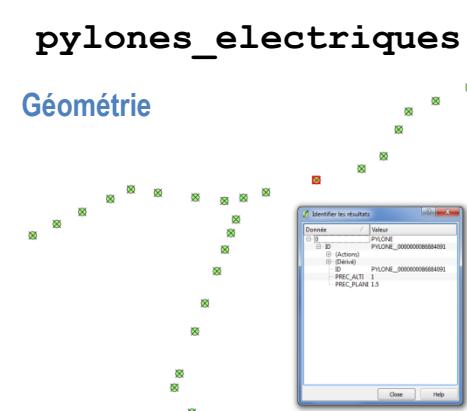
- Information géographique : données alphanumériques structurées (= attributs) et dotées de coordonnées géographiques (pour la localisation, = géométrie)
- Exemple 1 : observation de terrain (espece animal, poids, sexe, date, heure + coordonnées GPS)
- Exemple 2 : utilisation parcelle agricole (cultivateur, type culture, date début, date fin + contour parcelle)
- D'autres exemples ...



Attributs :  
numéro, nom, population, région ...



Attributs :  
Id troncon, nom voie



Attributs :  
Id pylone, hauteur, année construction

# Colonne géométrie 1/5 : principe

Consultation des données dans pgAdmin III

Fichier Édition Requêtes Favoris Macros Affichage Aide

Éditeur SQL Constructeur graphique de requêtes

Requêtes précédentes

```
SELECT code_dept, nom_dept, nom_chf, code_chf, nom_region, geom  
FROM public.geofla_departements  
ORDER BY code_dept;
```

Panneau sortie

Sortie de données Expliquer (Explain) Messages Historique

code_dept	nom_dept	nom_chf	code_chf	nom_region	geom
29	FINISTERE	QUIMPER	232	BRETAGNE	
2A	CORSE-DU-SUD	AJACCIO	004	CORSE	
30	HAUTE-CORSE	BASTIA	033	CORSE	
31	GARD	NIMES	189	Languedoc-Roussillon	
32	HAUTE-GARONNE	TOULOUSE	555	MIDI-PYRENEES	
33	GERS	AUCH	013	MIDI-PYRENEES	
34	GIRONDE	BORDEAUX	063	Aquitaine	
35	HERAULT	MONTPELLIER	172	Languedoc-Roussillon	
36	ILLE-ET-VILAINE	RENNES	238	BRETAGNE	
37	INDRE	CHATEAUROUX	044	CENTRE	
38	INDRE-ET-LOIRE	TOURS	261	CENTRE	
39	ISERE	GRENOBLE	185	Rhône-Alpes	
40	JURA	LONS-LE-SAUNIER	300	Franche-Comté	
41	LANDES	MONT-DE-MARSAN	192	Aquitaine	
42	LOIR-ET-CHER	BLOIS	018	CENTRE	
43	LOIRE	SAINTE-ETIENNE	218	Rhône-Alpes	
44	HAUTE-LOIRE	LE PUY-EN-VELAY	157	Auvergne	
45	LOIRE-ATLANTIQUE	NANTES	109	Pays de la Loire	
46	LOIRET	ORLEANS	234	CENTRE	
47	LOT	CAHORS	042	Midi-Pyrénées	
48	LOT-ET-GARONNE	AGEN	001	Aquitaine	
49	LOZERE	MENDE	095	Languedoc-Roussillon	
50	MAINE-ET-LOIRE	ANGERS	007	Pays de la Loire	
51	MANCHE	SAINTE-MARIE-DE	562	Basse-Normandie	

OK. Unix Ligne 1, Col 48, Caract. 48 96 lignes. 156 ms

Consultation des données dans QGIS

Remarque : dans pgAdmin III, la colonne géométrie peut sembler vide ...  
même lorsqu'elle contient des données !

# Colonne géométrie 2/5 : création d'une colonne géométrique

Création d'une table puis d'une colonne géométrie

- Dans PostGIS, la partie géométrique des données (points, polylignes, polygones géoréférencés) est enregistrée dans une **colonne géométrique**
- Une table peut avoir une ou plusieurs colonnes géométriques
- la fonction **AddGeometryColumn** permet d'ajouter une colonne géométrique à une table
  - préciser obligatoirement le **type de géométrie**, le nombre de dimensions (2 à 4) et le **système de coordonnées** utilisé
- La table de métadonnées **geometry\_columns** contient la liste des colonnes géométriques de la base

```
-- 1-creation attributs
CREATE TABLE public.bdt_48_communess
(
    code_insee character varying(5) NOT NULL,
    nom character varying(45),
    statut character varying(20),
    canton character varying(45),
    arrondisst character varying(45),
    popul integer,
    multican character varying(3),
    CONSTRAINT bdt_48_communess_pkey PRIMARY KEY (code_insee)
);

-- 2-creation colonne geometrie
SELECT AddGeometryColumn('public', 'bdt_48_communess',
    'the_geom', 2154, 'MULTIPOLYGON', 2);
```

Table geometry_columns								
oid	f_table_catalog	f_table_schema	f_table_name	f_geometry_type	coord_dimen	srid	type	geom_col
1	1521907	''	public.bdt_30_communess	geom	2	2154	MULTIPOLYGON	
2	1528030	''	public.bdt_48_communess	the_geom	2	2154	MULTIPOLYGON	
3	1522529	''	geofla_departements	geom	2	2154	MULTIPOLYGON	
*								

Annotations pointing to specific columns:

- Nom schema: points to the f\_table\_schema column.
- Nom table: points to the f\_table\_name column.
- Type géométrie: points to the f\_geometry\_type column.
- ID système coordonnées: points to the srid column.
- Nb dimensions: points to the coord\_dimen column.
- Nom colonne géométrique: points to the geom\_col column.

# Colonne géométrie 3/5 : les types géométriques supportés dans PostGIS

- Pour modéliser des phénomènes spatiaux en mode vectoriel, on utilise des points, lignes et polygones
- 6 principaux types géométriques sont utilisés dans PostGIS.
- 3 types simples : **POINT, LINESTRING, POLYGON**
- 3 types composés : **MULTIPOINT, MULTILINESTRING, MULTIPOLYGON**
- La notation **WKT** (Well Known Text) permet de les décrire de manière ‘compréhensible’

## Exemples de géométrie en notation WKT

Geometry primitives (2D)		
Type	Examples	
Point	POINT (30 10)	
LineString	LINESTRING (30 10, 10 30, 40 40)	
	POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10)) POLYGON ((35 10, 10 20, 15 40, 45 45, 35 10), (20 30, 35 35, 30 20, 20 30))	

Multipart geometries (2D)		
Type	Examples	
MultiPoint	MULTIPOINT ((10 40), (40 30), (20 20), (30 10)) MULTIPOINT (10 40, 40 30, 20 20, 30 10)	
	MULTILINESTRING ((10 10, 20 20, 10 40), (40 40, 30 30, 40 20, 30 10))	
MultiPolygon	MULTIPOLYGON (((30 20, 10 40, 45 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5))) MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)), ((20 35, 45 20, 30 5, 10 10, 10 30, 20 35), (30 20, 20 25, 20 15, 30 20)))	
		

[http://en.wikipedia.org/wiki/Well-known\\_text](http://en.wikipedia.org/wiki/Well-known_text)

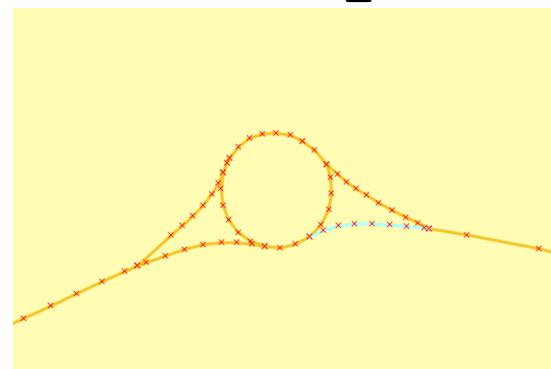
# Colonne géométrie 4/5 : identifions ces types géométriques ...

departements



MULTIPOLYGON

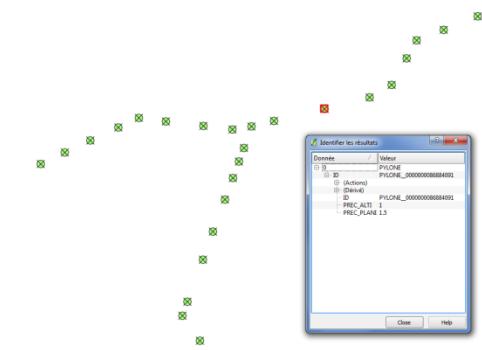
troncons\_route



LINESTRING

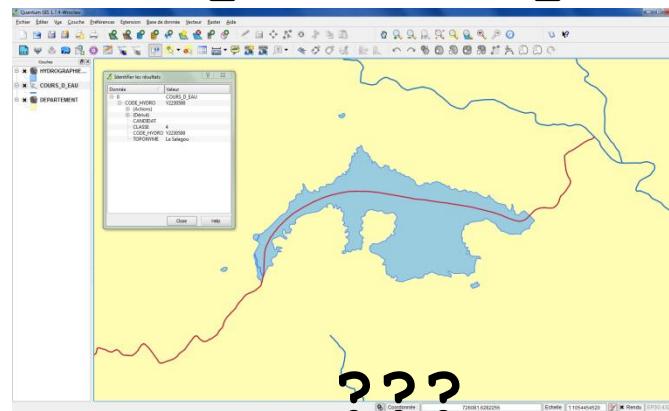
... ou MULTILINESTRING

pylones\_electriques



POINT

troncons\_hydro, surface\_hydro



# Colonne géométrie 5/5 :

## Les systèmes de coordonnées dans PostGIS

- Dans de nombreux SIG, les identifiants **EPSG** permettent de désigner les **systèmes de coordonnées**
- Les SRID (**Spatial Reference System Identifier**) répertoriés dans la table *spatial\_ref\_sys* correspondent aux codes EPSG ... en général

Dénomination ArcGIS	EPSG
RGF Lambert 93	<b>2154</b>
GCS WGS 84	<b>4326</b>
NTF Lambert Zone 3	<b>27563</b>
NTF Lambert 2 étendu	<b>27572</b>
NTF Lambert 3 Carto	<b>27573</b>
UTM 30N WGS 84	<b>32360</b>
UTM 31N WGS 84	<b>32361</b>
UTM 32N WGS 84	<b>32362</b>

<http://spatialreference.org/>

**SRID PostGIS** Outils Aide Définition du SC (format WKT)

srid [PK] integer	auth_name character varying	auth_srid integer	srtext character varying(2048)
148	2147	2147	PROJCS["NAD83(CSRS98) / UTM zone 10 (deprecated)",GEOGCS["NAD83(CSRS98)",proj=utm+zone=10+ellps=GRS80+towgs84=0,0,0,0,0,0,0+units=m+no_defs],PROJCS["NAD83(CSRS98) / UTM zone 21N (deprecated)",GEOGCS["NAD83(CSRS98)",proj=utm+zone=21+ellps=GRS80+towgs84=0,0,0,0,0,0,0+units=m+no_defs],PROJCS["NAD83(CSRS98) / UTM zone 18N (deprecated)",GEOGCS["NAD83(CSRS98)",proj=utm+zone=18+ellps=GRS80+towgs84=0,0,0,0,0,0,0+units=m+no_defs],PROJCS["NAD83(CSRS98) / UTM zone 17N (deprecated)",GEOGCS["NAD83(CSRS98)",proj=utm+zone=17+ellps=GRS80+towgs84=0,0,0,0,0,0,0+units=m+no_defs],PROJCS["NAD83(CSRS98) / UTM zone 13N (deprecated)",GEOGCS["NAD83(CSRS98)",proj=utm+zone=13+ellps=GRS80+towgs84=0,0,0,0,0,0,0+units=m+no_defs],PROJCS["NAD83(CSRS98) / UTM zone 12N (deprecated)",GEOGCS["NAD83(CSRS98)",proj=utm+zone=12+ellps=GRS80+towgs84=0,0,0,0,0,0,0+units=m+no_defs],PROJCS["NAD83(CSRS98) / UTM zone 11N (deprecated)",GEOGCS["NAD83(CSRS98)",proj=utm+zone=11+ellps=GRS80+towgs84=0,0,0,0,0,0,0+units=m+no_defs],PROJCS["RGF93 / Lambert-93",GEOGCS["RGF93",DATUM["Reseau_Geodesique_France_1993",proj=lcc+lat_1=49+lat_2=44+lat_0=-46.5+lon_0=3+x_0=700000+y_0=6600000+units=m+no_defs],PROJCS["American Samoa 1962 / American Samoa Lambert (deprecated)",GEOGCS["AMerican Samoa 1962",proj=lcc+lat_1=-14.26666666666666+lat_0=-14.26666666666666+units=m+no_defs],PROJCS["NAD83(HARN) / UTM zone 59S (deprecated)",GEOGCS["NAD83(HARN)",proj=utm+zone=59+south+ellps=GRS80+units=m+no_defs],PROJCS["IRENET95 / Irish Transverse Mercator",GEOGCS["IRENET95",DATUM["IRENET95",proj=tmerc+lat_0=53.5+lon_0=-8+k=0.99982+x_0=600000+y_0=6666666666666666+units=m+no_defs],PROJCS["IRENET95 / UTM zone 29N",GEOGCS["IRENET95",DATUM["IRENET95",proj=utm+zone=29+ellps=GRS80+towgs84=0,0,0,0,0,0,0+units=m+no_defs],PROJCS["Sierra Leone 1924 / New Colony Grid",GEOGCS["Sierra Leone 1924",proj=tmerc+lat_0=6.666666666666667+lon_0=-12+k=1+x_0=0+y_0=0],PROJCS["Sierra Leone 1924 / New War Office Grid",GEOGCS["Sierra Leone 1924",proj=tmerc+lat_0=6.666666666666667+lon_0=-12+k=1+x_0=0+y_0=0],PROJCS["Sierra Leone 1968 / UTM zone 28N",GEOGCS["Sierra Leone 1968",proj=utm+zone=28+ellps=GRS80+towgs84=-88.4101+n_0=0+e_0=0],PROJCS["Sierra Leone 1968 / UTM zone 29N",GEOGCS["Sierra Leone 1968",proj=utm+zone=29+ellps=GRS80+towgs84=-88.4101+n_0=0+e_0=0]]]]]

3749 lignes.

**Définition du SC (format PROJ4)**

Table spatial\_ref\_sys

# Définir une géométrie avec les fonctions de construction géométrique

- **WKT / WKB** : la notation **Well-Known Text** (*humainement lisible*) est complétée par la notation **Well-Known Binary** qui permet de ‘condenser’ sous une forme binaire les géométries
- Deux principales fonctions permettent de construire des géométries :  
**ST\_GeomFromText** et **ST\_GeomFromWKB**

... il existe bien d'autres fonctions PostGIS dans la catégorie  
*Geometry Constructors* pour définir des géométries ...

```
-- 4 manières de définir le point de 43,6381°N et 3,8639°E (WGS84)
SELECT ST_GeomFromText('POINT(3.8639 43.6381)', 4326) AS the_geomtest;
SELECT ST_SetSRID(ST_GeomFromText('POINT(3.8639 43.6381)'), 4326) AS the_geomtest;
SELECT ST_SetSRID(ST_Point(3.8639, 43.6381), 4326) AS the_geomtest;
SELECT ST_GeomFromWKB(E'\x001\x001\x000\x000~\x0358gD\x351\x016@\x357\x311
\x303B\x255\x321E@', 4326) AS the_geomtest;
```

## Name

ST\_GeomFromText — Return a specified ST\_Geometry value from Well-Known Text representation (WKT).

## Synopsis

```
geometry ST_GeomFromText(text WKT);
geometry ST_GeomFromText(text WKT, integer srid);
```

## Description

Constructs a PostGIS ST\_Geometry object from the OGC Well-Known text representation.

<http://postgis.refractions.net/documentation/manual-1.5/reference.html>



# Colonne géométrie 6/6 : contraintes spatiales

- Remarque : l'appel de la fonction **AddGeometryColumn** induit la création de 3 contraintes :
  - Sur le type de géométrie
  - Sur le nombre de dimensions
  - Sur le SRID

Les géométries insérées dans la colonne **the\_geom** devront être :

- de type **MULTIPOLYGON**, ou être vides (**NULL**)
- définies par des points à 2D (X Y)
- définies par des coordonnées en Lambert 93  
(le SRID 2154 devra être explicitement indiqué pour chaque géométrie)

```
SELECT AddGeometryColumn('public',
'bdt_48_communes', 'the_geom', 2154,
'MULTIPOLYGON', 2);
```

The screenshot shows the pgAdmin interface. At the top, a query window displays the SQL command to add a geometry column:

```
SELECT AddGeometryColumn('public',
'bdt_48_communes', 'the_geom', 2154,
'MULTIPOLYGON', 2);
```

A red arrow points from this query down to the pgAdmin interface. In the interface, the left pane shows the database structure:

- Tables: bdt\_30\_communes, bdt\_48\_communes, geofla\_departements
- Views: None
- Functions: None
- Procedures: None
- Sequences: None
- Triggers: None
- Indices: None
- Rules: None
- Constraints: bdt\_48\_comunes\_pkey, enforce\_dims\_the\_geom, enforce\_geotype\_the\_geom, enforce\_srid\_the\_geom
- Indexes: None
- Triggers: None
- Views: None

A red box highlights the 'Contraintes' section, which contains the four spatial constraints. A red bracket on the left groups the first three items under 'de type MULTIPOLYGON, ou être vides (NULL)'. Another red bracket groups the last three items under 'définies par des coordonnées en Lambert 93 (le SRID 2154 devra être explicitement indiqué pour chaque géométrie)'.

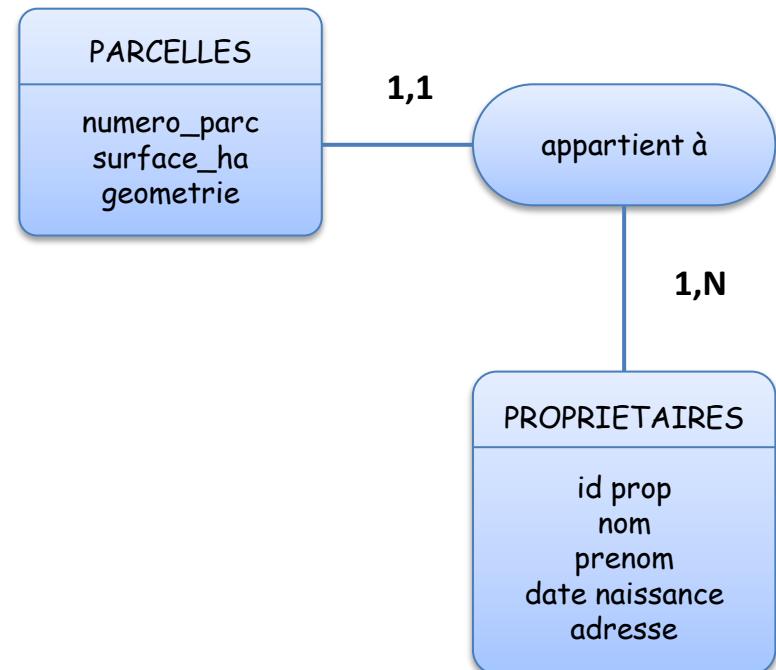
At the bottom, the table definition is shown:

```
CREATE TABLE bdt_48_communes
(
    gid serial NOT NULL,
    nom character varying(45),
    code_insee character varying(5),
    statut character varying(20),
    canton character varying(45),
    arrondisst character varying(45),
    popul integer,
    multican character varying(3),
    the_geom geometry,
    CONSTRAINT bdt_48_comunes_pk PRIMARY KEY (gid)
)
CONSTRAINT enforce_dims_the_geom CHECK (st_ndims(the_geom) = 2),
CONSTRAINT enforce_geotype_the_geom CHECK (geometrytype(the_geom) = 'MULTIPOLYGON'::text OR the_geom IS NULL),
CONSTRAINT enforce_srid_the_geom CHECK (st_srid(the_geom) = 2154)
```

# Exemple : mise à place d'une BD Parcelles

## Modèle conceptuel de données

- Le modèle conceptuel de données comporte 2 entités :
  - Parcelles** (numéro, surface ha, géométrie)
  - Propriétaires** (id, nom, prénom, date naissance, adresse)
- Relation entre les 2 entités :
  - Un propriétaire peut posséder **plusieurs parcelles**
  - Une parcelle appartient à **un seul propriétaire**



# Exemple : mise à place d'une BD Parcelles

## Création des tables

```
-- creation table proprietaires
--  

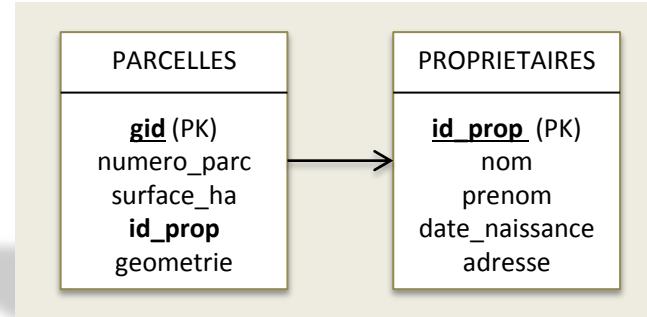
CREATE TABLE proprietaires (
    id_prop integer NOT NULL,
    nom character varying(50),
    prenom character varying(50),
    date_naissance date,
    adresse character varying(200)
);
-- ajout cle primaire
ALTER TABLE proprietaires ADD CONSTRAINT proprietaires_pkey PRIMARY KEY (id_prop);  

--  

-- creation table parcelles
--  

CREATE TABLE parcelles (
    gid serial NOT NULL,  

    numero_parc character varying(5),
    surface_ha double precision,
    id_prop integer
);
-- ajout cle primaire
ALTER TABLE parcelles ADD CONSTRAINT parcelles_pkey PRIMARY KEY (gid);
-- ajout cle etrangere
ALTER TABLE parcelles ADD CONSTRAINT parcelles_fkey_prop FOREIGN KEY (id_prop)
    REFERENCES proprietaires (id_prop) MATCH SIMPLE;
-- ajout colonne geometrique :
SELECT AddGeometryColumn('public', 'parcelles', 'geometrie', 2154, 'POLYGON', 2);
```



Choix d'une **clé primaire de type entier** et auto-incrémente : pour l'affichage dans QGIS

Définition de la relation parcelles-propriétaire avec une **clé étrangère**

Création de la **colonne géométrique** *geometrie* : POLYGON 2D, Coordonnées Lambert 93 (EPSG:2154)

# Exemple : mise à place d'une BD Parcelles

## Insertion de données

```
-- inserer des donnees proprietaires
INSERT INTO proprietaires(id_prop, nom, prenom, date_naissance, adresse)
VALUES (1, 'DUPONT', 'Jean', '05/04/1963', '8 rue de la Tour 34000 MONTPELLIER');
INSERT INTO proprietaires(id_prop, nom, prenom, date_naissance, adresse)
VALUES (2, 'DUBOIS', 'Pierre', '06/05/1964', '12 chemin du Puit 34500 BEZIERS');
INSERT INTO proprietaires(id_prop, nom, prenom, date_naissance, adresse)
VALUES (3, 'DURAND', 'Albert', '29/02/1960', '17 rue de la Monnaie 34700 LODEVE');

-- inserer des donnees parcelles
INSERT INTO parcelles(numero_parc, surface_ha, id_prop, the_geom)
VALUES (
    '129', 0.0, 1,
    ST_GeomFromText('POLYGON((741137 6288650,741201 6288799,741257 6288776,741243 6288735,741222
        6288743,741201 6288694,741176 6288636,741137 6288650))', 2154)
);
INSERT INTO parcelles(numero_parc, surface_ha, id_prop, the_geom)
VALUES (
    '130', 0.0, 2,
    ST_GeomFromText('POLYGON((741201 6288694,741222 6288743,741243 6288735,741292 6288714,741282
        6288686,741270 6288664,741201 6288694))', 2154)
);
INSERT INTO parcelles(numero_parc, surface_ha, id_prop, the_geom)
VALUES (
    '131', 0.0, 3,
    ST_GeomFromText('POLYGON((741176 6288636,741201 6288694,741270 6288664,741245 6288609,741176
        6288636))', 2154)
);
```

Définition de la **géométrie** (polygone parcelle)  
à insérer dans la colonne *the\_geom*

# Exemple: mise à place d'une BD Parcelles

Quantum GIS 1.7.4-Wroclaw - digit\_parcelles

Fichier Éditer Vue Couche Préférences Extension Base de données Vecteur Raster Aide

Couches

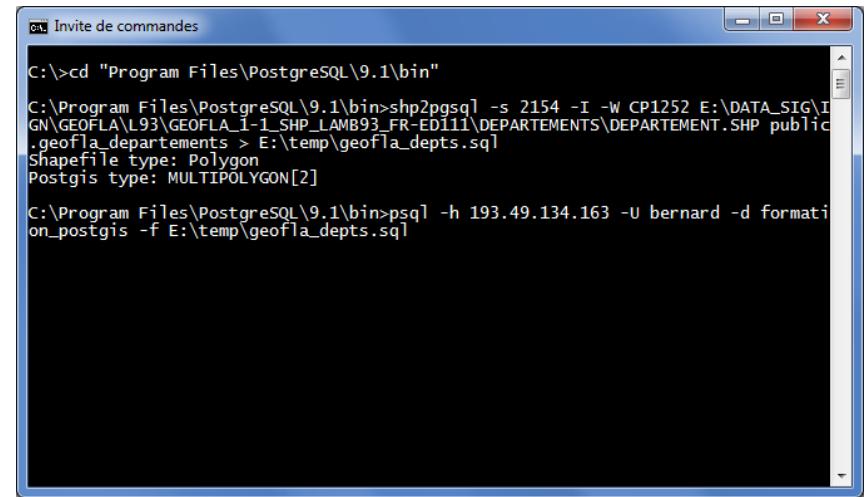
- bornes\_parcelles\_xy
- parcelles
- 34-2009-0740-6290-LA93-C07

```
INSERT INTO parcelles(numero_parc, surface_ha, id_prop, the_geom)
VALUES (
    '129', 0.0, 1,
    ST_GeomFromText('POLYGON((741137 6288650, 741201 6288799, 741257 6288776, 741243 6288735, 741222 6288743, 741201 6288694, 741176 6288636, 741137 6288650))', 2154)
);
```

```
INSERT INTO parcelles(numero_parc, surface_ha, id_prop, the_geom)
VALUES (
    '132', 0.0, 1,
    ST_GeomFromText('POLYGON((741245 6288609, 741276 6288593, 741248 628850, 741245 6288609))',
    2154)
);
```

# Importer des shapefiles dans PostGIS : shp2pgsql

- shp2pgsql : l'utilitaire 'de base' pour importer des données dans PostGIS
- Principe : **shp2pgsql** convertit un Shapefile en un fichier SQL qui permettra de récréer les données dans PostGIS
- Syntaxe de la ligne de commande :  
**shp2pgsql options data\_in.shp**  
nomschema.nomtable **data\_out.sql**
- Options les plus courantes
  - s SRID (définition du SRID)
  - I (création d'un index spatial)
  - W ENCODAGE (encodage du shapefile : CP1252)



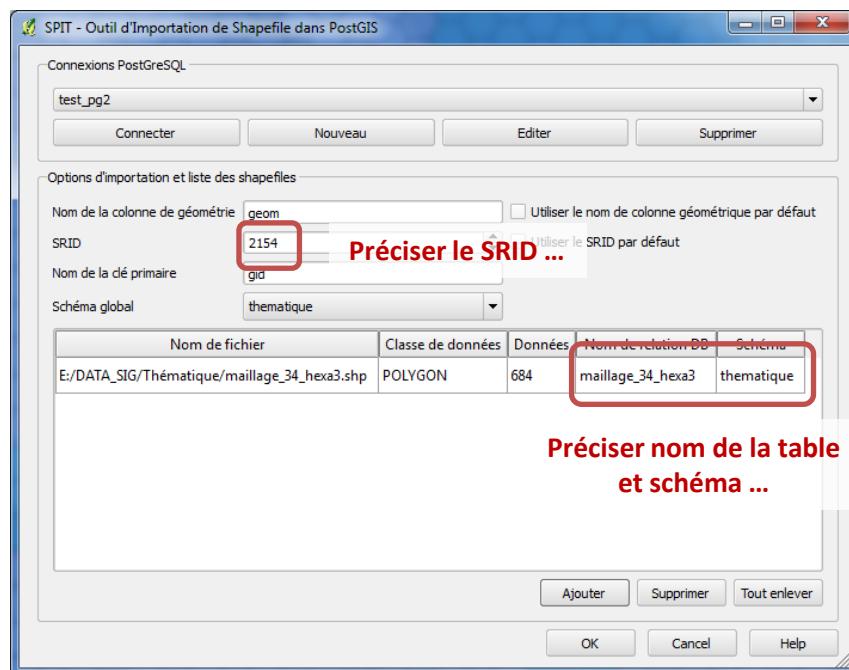
```
c:\ Invité de commandes
C:\>cd "Program Files\PostgreSQL\9.1\bin"
C:\Program Files\PostgreSQL\9.1\bin>shp2pgsql -s 2154 -I -W CP1252 E:\DATA_SIG\IGN\GEOFLA\L93\GEOFLA_1_1_SHP_LAMB93_FR-ED111\DEPARTEMENTS\DEPARTEMENT.SHP public.geofla_departements > E:\temp\geofla_depts.sql
Shapefile type: Polygon
Postgis type: MULTIPOLYGON[2]
C:\Program Files\PostgreSQL\9.1\bin>psql -h 193.49.134.163 -U bernard -d formation_postgis -f E:\temp\geofla_depts.sql
```

```
D:\DATA>shp2pgsql -s 2154 -I -W CP1252
COMMUNES.shp public.bdt_34_communes >
communes_34.sql
```

```
D:\DATA>psql -U postgres -d bd_formation -f
communes_34.sql
```

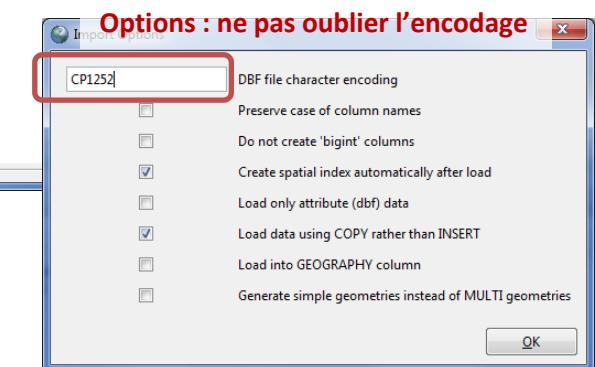
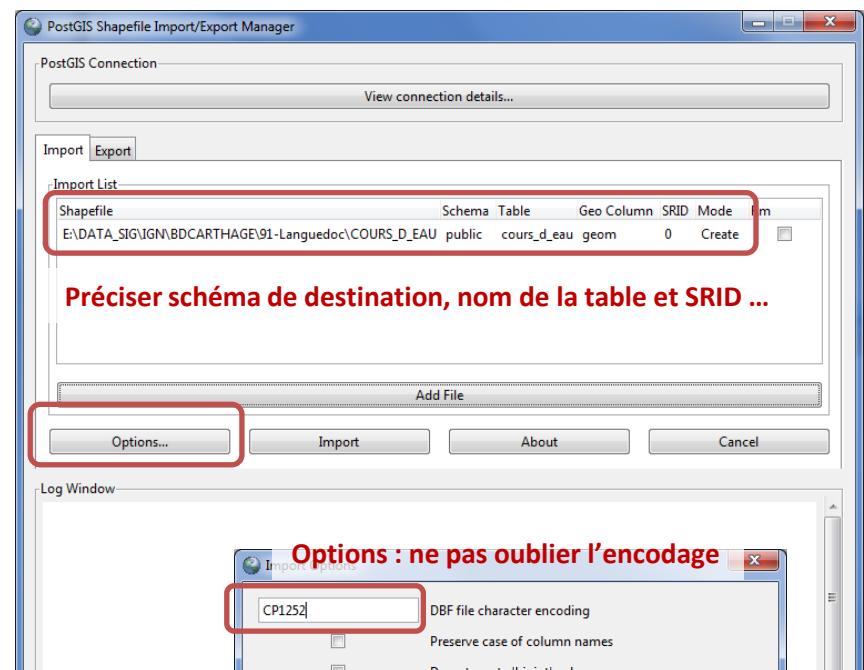
# Importer des shapefiles dans PostGIS : par interface graphique

## SPIT (extension de QGIS)



Possibilité d'importer par lot plusieurs shapefiles ...

## shp2pgsql-gui



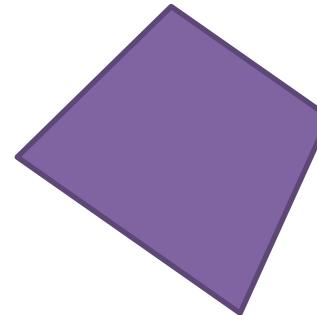
Formation PostGIS 1.5

# **MESURES DE LONGUEUR ET DE SURFACE**

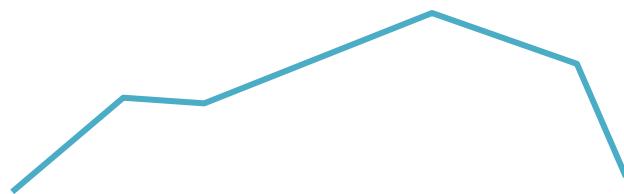
# ST\_Area, ST\_Length

- PostGIS dispose de deux fonctions ‘classiques’ pour mesurer respectivement les longueurs et surfaces : **ST\_Length** et **ST\_Area**
- Elles prennent en argument la **géométrie à mesurer**
- L’unité de mesure est basée sur l’unité du système de coordonnées
  - Lambert 93 → mètre pour les longueurs, mètre carré pour les surfaces
- Autres fonctions similaires : **ST\_Perimeter**

`ST_Area(geometrie)`



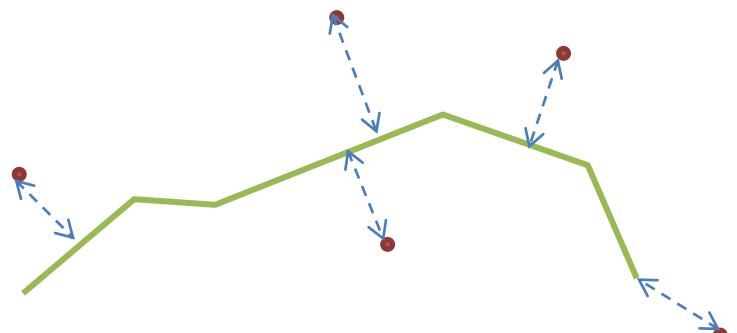
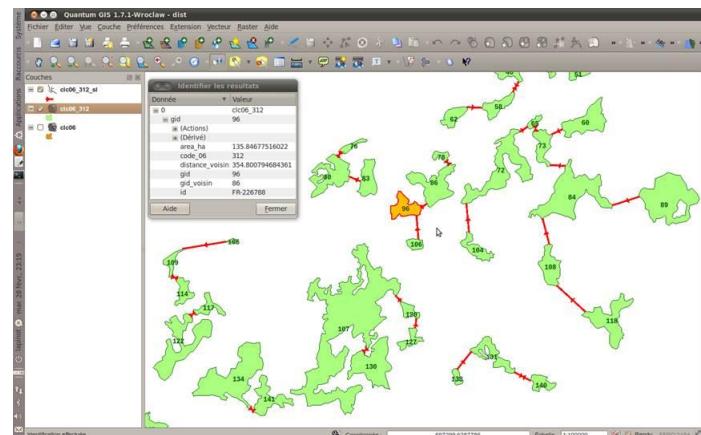
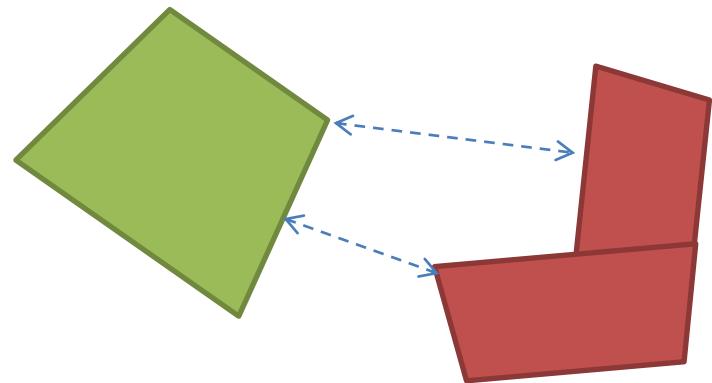
`ST_Length(geometrie)`



# Mesure de distance entre entités

- PostGIS dispose également d'une fonction pour mesurer la distance minimale entre 2 géométries : **ST\_Distance**
  - Pour déterminer la **distance de bord à bord** entre **2 polygones**
  - Et aussi : **distance d'un point au segment le plus proche d'une polyligne**
  - Et bien sûr, **distance d'un point à un autre point**
- Remarque : **ST\_ShortestLine** pour matérialiser le plus court trajet

`ST_Distance(geomtrieA, geomtrieB)`



[http://postgis.refractions.net/documentation/manual-1.5/ST\\_Distance.html](http://postgis.refractions.net/documentation/manual-1.5/ST_Distance.html)

# ST\_Area : exemple de calcul de surface

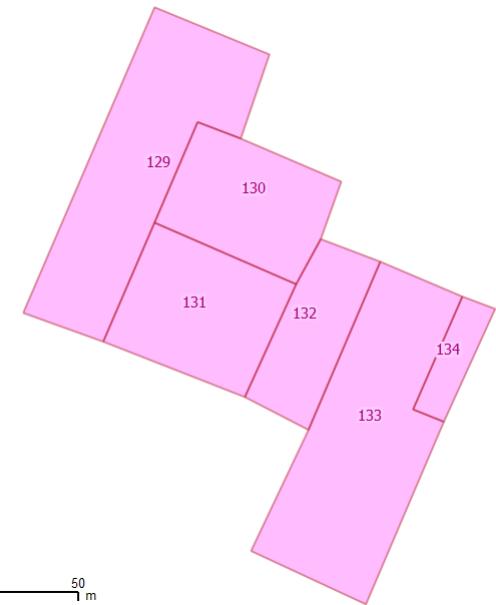
```
SELECT numero_parc, surf_ha  
FROM parcelles;
```

Sortie de données		Expliquer (Explain)
	numero_parc	surf_ha
	character varying	double precision
1	129	0
2	130	0
3	131	0
4	132	0
5	133	0
6	134	0

1. Nous partons d'une table avec un champ surf\_ha non calculé

```
UPDATE parcelles  
SET surf_ha = ST_Area(the_geom) / 10000;
```

Sortie de données				Expliquer (Explain)	Messages	Historique
La requête a été exécutée avec succès : 6 lignes modifiées. La requête a été exécutée en 21 ms.						



3. Voir le résultat du calcul

```
SELECT numero_parc, surf_ha  
FROM parcelles;
```

Sortie de données			Expliquer (Explain)	Messages
	numero_parc	surf_ha		
	character varying(5)	double precision		
1	129	0.75815		
2	130	0.4157		
3	131	0.4611		
4	132	0.2926		
5	133	0.8325		
6	134	0.09995		

# SQL avancé : un déclencheur pour garder à jour le champ *surf\_ha*

- Limitation de l'exemple précédent : le champ *surf\_ha* n'est pas mis à jour automatiquement en cas de **création ou modification d'une nouvelle géométrie**



- Pour calculer automatiquement la surface à la création / modification d'une parcelle, implémenter un déclencheur (**trigger**)

```
CREATE OR REPLACE FUNCTION calc_surf_parcelle()
RETURNS TRIGGER AS $maj_surf_parcelle$
BEGIN
    NEW.surf_ha := ST_Area(NEW.the_geom) /
10000;
    RETURN NEW;
END;
$maj_surf_parcelle$ LANGUAGE plpgsql;

CREATE TRIGGER maj_surf_parcelle
BEFORE INSERT OR UPDATE OF the_geom ON
parcelles
FOR EACH ROW
EXECUTE PROCEDURE calc_surf_parcelle();
```

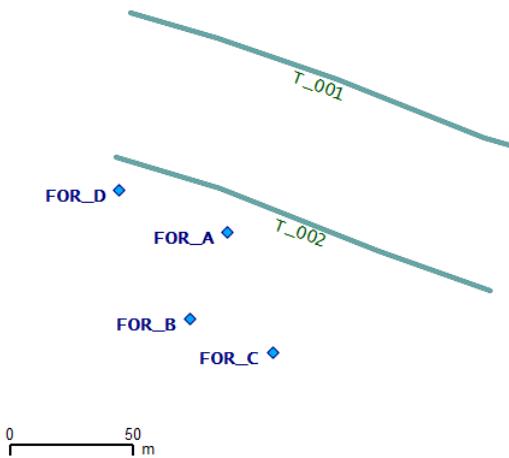
# ST\_Distance : exemple

```
SELECT f.nom_forage, c.id tuyau,  
       ST_Distance(f.the_geom, d.the_geom) AS distance_m  
FROM forages AS f, conduites AS c;
```

nom_forage character varying(80)	id tuyau character varying(40)	distance_m double precision
FOR A	T 001	73.0668740590
FOR B	T 001	111.225625801
FOR C	T 001	112.524120162
FOR D	T 001	70.2589516871
FOR A	T 002	15.3414108952
FOR B	T 002	53.5924179614
FOR C	T 002	53.9256085090
FOR D	T 002	12.3438564409

Distance minimale entre forages  
(POINT) et conduites (LINESTRING)

Cette requête effectue le produit  
cartésien des 2 tables : toutes les  
combinaisons sont calculées.



Formation PostGIS 1.5

# **SYSTEMES DE COORDONNEES TYPE GEOGRAPHY**

# Transformation de coordonnées (passage d'un système à un autre)

- Il est possible de créer dans une même table plusieurs colonnes géométrie avec différents systèmes de coordonnées
- Rappel : au moment de sa définition, on précise toujours le système de coordonnées d'un géométrie
- La fonction **ST\_Transform**(geometrie, srid) permet de « convertir » une géométrie dans un autre système

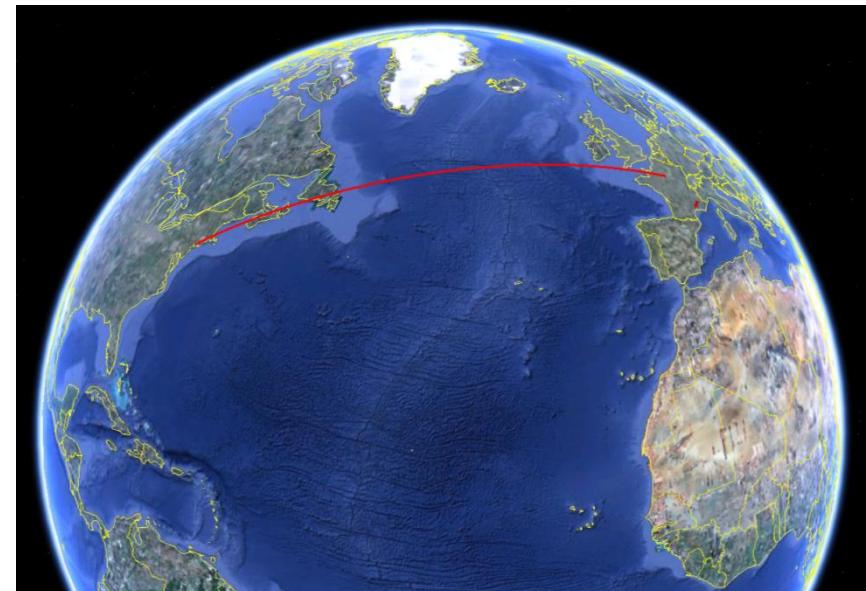
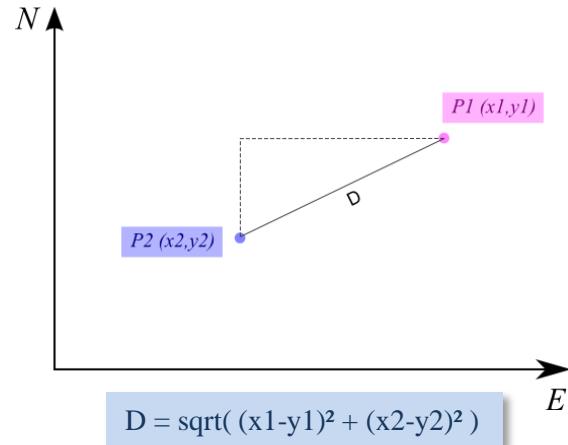
```
CREATE TABLE releves (
    gid serial,
    nom_lieu character varying(40),
);
-- ajout colonne geom WGS84
SELECT AddGeometryColumn('public', 'releves',
'geom_wgs84', 4326, 'POINT', 2);
-- insérer un point
INSERT INTO releves(nom_lieu, geom_wgs84)
VALUES ('salle formation',
ST_GeomFromText('POINT (3.8639 43.6381)', 4326)
);

-- ajout colonne geom Lambert 93
SELECT AddGeometryColumn('public', 'releves',
'geom_193', 2154, 'POINT', 2);
-- convertir WGS84 -> Lambert93
UPDATE releves
SET geom_193 = ST_Transform(geom_wgs84, 2154);
-- afficher coord 193
SELECT gid, nom_lieu, ST_X(geom_193) AS coordx,
ST_Y(geom_193) AS coordy
FROM releves;
```

gid	nom_lieu	coordx	coordy
integer	character varying(40)	double precision	double precision
1	salle formation	769728.727324	6282498.63949

# Mesures de distance avec le type geometry et le type geography

- Avec le **type geometry**, les distances sont calculées avec le théorème de Pythagore dans l'unité de mesure du SRID
- Conséquence : pour les géométries avec la SRID 4326 (GCS WGS 84), PostGIS calculera des longueurs en degrés > mesures **non-significatives**
- Le **type geography** va nous permettre de mesurer des distances géodésiques (**arc entre deux points sur la surface de l'ellipsoïde WGS 84**), à partir de coordonnées en degrés décimaux
- Plus d'info : voir exemple  
<http://www.postgis.fr/chrome/site/docs/workshop-foss4g/doc/geography.html>



```
> dd <- function (d,m,s) { return(((s/60)+m)/60)+d }  
> dd(40,44,54)  
[1] 40.74833  
> dd(73,59,11)  
[1] 73.98639  
> dd(48,51,30)  
[1] 48.85833  
> dd(2,17,40)  
[1] 2.294444
```

# Le type geography, exemple

- Pour ajouter une colonne géographique dans une table :

```
ALTER TABLE releves ADD COLUMN  
the_geog geography(POINT, 4326);
```

Table releves		
Nom de colonne	Définition	Hérité ...
gid	serial NOT NULL	
nom_lieu	character varying(40)	
geom_wgs84	geometry	
geom_93	geometry	
the_geog	geography(Point,4326)	

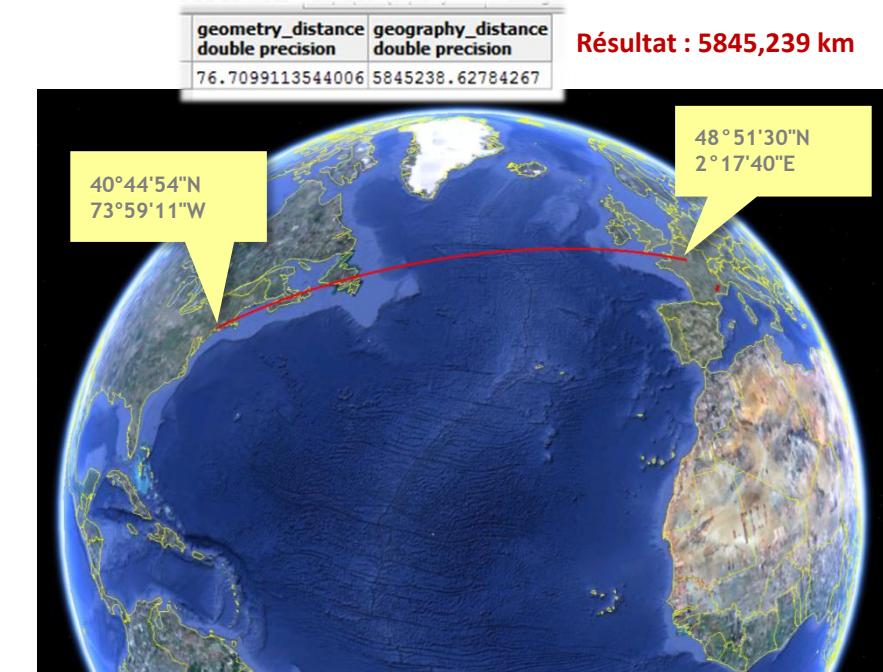
- Pour convertir une géométrie dans le type geography, utiliser la fonction **Geography(geometry)** :

```
UPDATE releves  
SET the_geog = Geography(geometry);
```

- Le constructeur **GeographyFromText(text)** permet de définir par exemple un point GPS dans le type *geography*.

Exemple : mesurer distance Paris/New-York à partir des coordonnées WGS84

```
SELECT ST_Distance(  
    ST_GeometryFromText('Point(2.294 48.858)', 4326),  
    ST_GeometryFromText('Point(-73.986 40.748)', 4326)  
) AS geometry_distance,  
ST_Distance(  
    ST_GeographyFromText('Point(2.294 48.858)'),  
    ST_GeographyFromText('Point(-73.986 40.748)')  
) AS geography_distance;
```

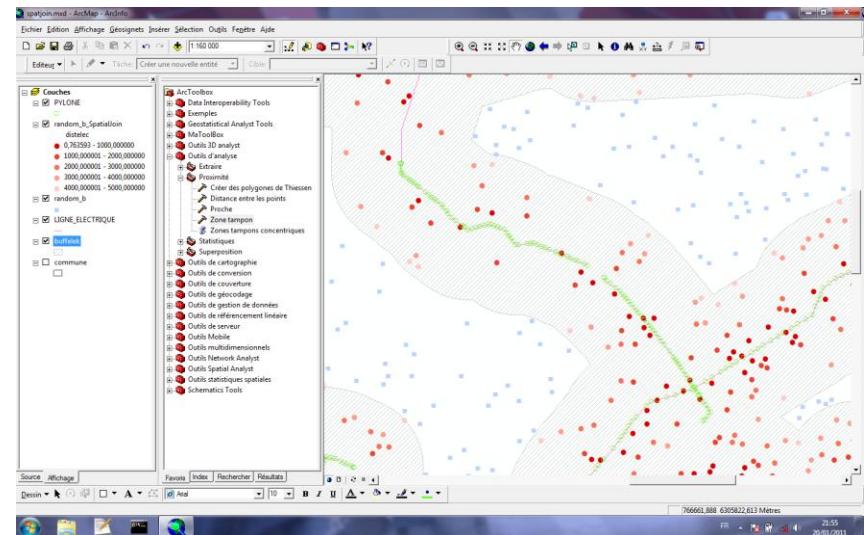


Formation PostGIS 1.5

# **REQUETES SPATIALES**

# Qu'est-ce qu'une requête spatiale ?

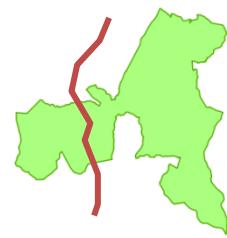
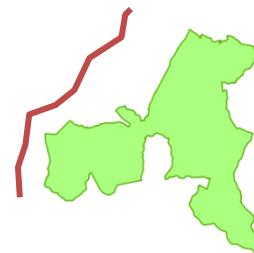
- C'est une requête qui porte sur les **relations spatiales** entre 2 types d'entités
- Exemple 1 : chercher les communes traversées par la ligne à haute tension
- Exemple 2 : chercher les pylônes dans un rayon de 100 m autour des routes



# Les fonctions de relation spatiale

- On utilise des fonctions spatiales de type « relation spatiale (A, B) » qui renvoie **VRAI** ou **FAUX**
- Généralement ces fonctions prennent 2 géométries en argument (parfois un argument distance en plus)
- Quelques relations spatiales ...
  - A intersecte B
  - A est compris dans un rayon de n mètres autour de B
  - A est contenu dans B
  - A contient B
  - A touche B
  - A est identique à B

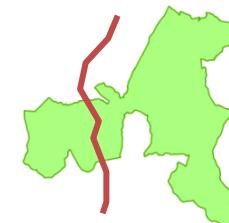
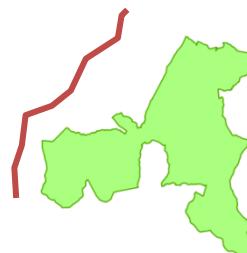
- **A intersecte B ?**



FAUX

VRAI

- **A est contenu dans B ?**



FAUX

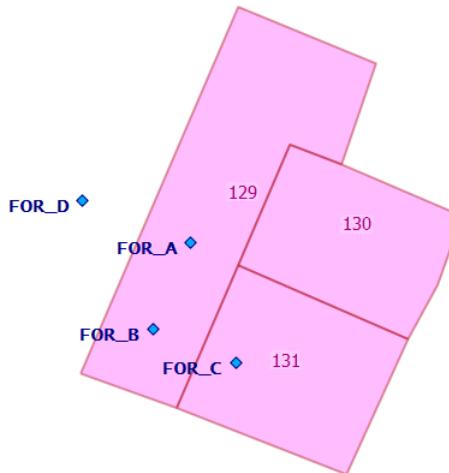


VRAI

# Exemple : la fonction ST\_Intersects

- La fonction **ST\_Intersects** prend 2 géométrie en argument
- [http://postgis.refractions.net/documentation/manual-1.5/ST\\_Intersects.html](http://postgis.refractions.net/documentation/manual-1.5/ST_Intersects.html)
- **Exemple** : sélectionner les points de forage contenus dans les parcelles ...
- ... et déterminer dans quelle parcelle est située chaque point

```
SELECT f.nom_forage, p.numero_parc  
FROM forages AS f, parcelles AS p  
WHERE ST_Intersects(f.geom, p.geom) ;
```



Éditeur SQL Constructeur graphique de requêtes

Requêtes précédentes Supprimer Tout supprimer

```
SELECT f.nom_forage, p.numero_parc  
FROM forages AS f, parcelles AS p  
WHERE ST_Intersects(f.the_geom, p.the_geom) ;
```

Panneau sortie

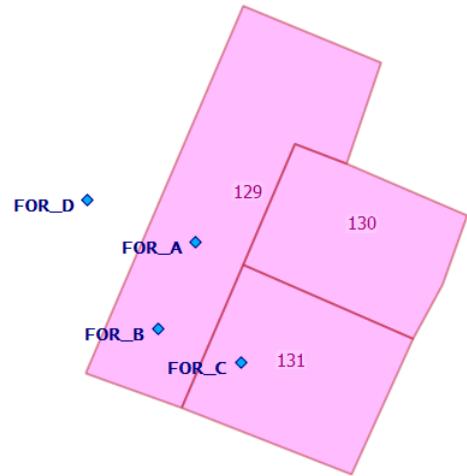
	nom_forage character varying(80)	numero_parc character varying(5)
1	FOR A	129
2	FOR B	129
3	FOR C	131

# Fonctionnement de la fonction ST\_Intersects dans la clause WHERE

```
SELECT f.nom_forage, p.numero_parc  
FROM forages AS f, parcelles AS p;
```

nom_forage character var	numero_parc character vary
FOR A	129
FOR B	129
FOR C	129
FOR D	129
FOR A	130
FOR B	130
FOR C	130
FOR D	130
FOR A	131
FOR B	131
FOR C	131
FOR D	131

1. Produit cartésien des 2 tables !



```
SELECT f.nom_forage, p.numero_parc,  
ST_Intersects(f.the_geom, p.the_geom) AS test_inter  
FROM forages AS f, parcelles AS p;
```

nom_forage character var	numero_parc character vary	test_inter boolean
FOR A	129	t
FOR B	129	t
FOR C	129	f
FOR D	129	f
FOR A	130	f
FOR B	130	f
FOR C	130	f
FOR D	130	f
FOR A	131	f
FOR B	131	f
FOR C	131	t
FOR D	131	f

2. Produit cartésien avec test d'intersection

nom_forage character var	numero_parc character vary
FOR A	129
FOR B	129
FOR C	131

3. La clause WHERE agit comme un crible pour ne conserver que les lignes qui respectent le critère

```
SELECT f.nom_forage, p.numero_parc  
FROM forages AS f, parcelles AS p  
WHERE (ST_Intersects(f.the_geom, p.the_geom));
```

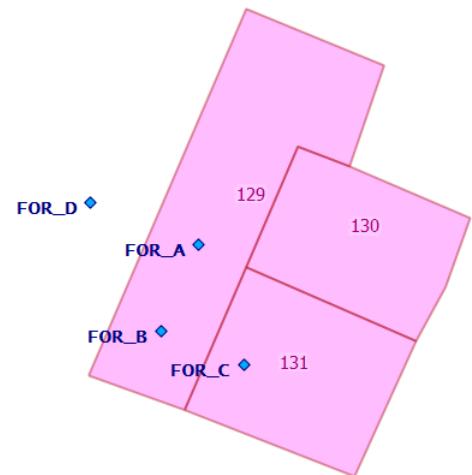
# SQL avancé : une jointure externe gauche (LEFT OUTER JOIN)

Nous cherchons la liste *exhaustive* des forages, avec la parcelle contenant le forage (*si elle existe ...*)

```
SELECT f.nom_forage, p.numero_parc
FROM forages AS f LEFT OUTER JOIN parcelles AS p
ON (ST_Intersects(f.the_geom, p.the_geom));
```

Jointure externe gauche avec  
utilisation de la fonction  
`ST_Intersects`

nom_forage character varying(80)	numero_parc character varying(5)
FOR A	129
FOR B	129
FOR C	131
FOR D	



# Clause GROUP BY et fonctions d'agrégation

```
SELECT p.numero_parc, f.nom_forage, f.profondeur_m
FROM forages AS f, parcelles AS p
WHERE ST_Intersects(f.the_geom, p.the_geom) ;;
```

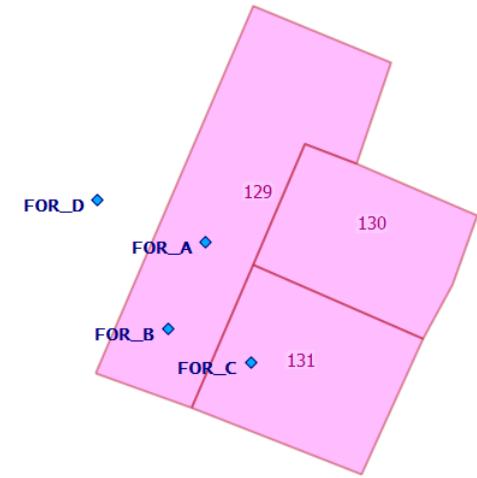
numero_parc character vary	nom_forage character var	profondeur_m numeric(5,2)
129	FOR A	10.00
129	FOR B	15.00
131	FOR C	11.50

1. Liste des forages avec leur profondeur

```
SELECT p.numero_parc, count(f.nom_forage) AS nb_forages,
       avg(f.profondeur_m) AS moy_prof_m
  FROM forages AS f, parcelles AS p
 WHERE ST_Intersects(f.the_geom, p.the_geom)
 GROUP BY p.numero_parc;
```

numero_parc character vary	nb_forages bigint	moy_prof_m numeric
129	2	12.500000000000000
131	1	11.500000000000000

2. Chercher le nb de forages par parcelle et la profondeur moyenne



Pour arrondir la colonne moy\_prof\_m, « caster » le résultat de avg :

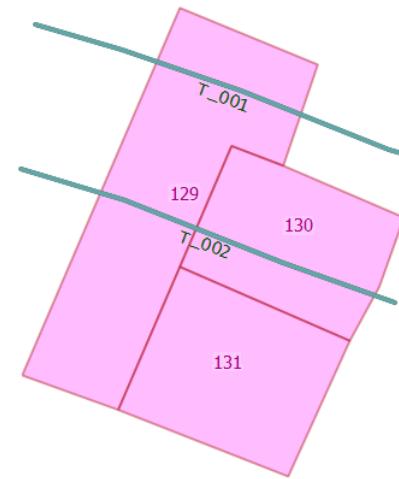
`avg(f.profondeur_m)::numeric(5,2) AS moy_prof_m`

# Clause GROUP BY ... autre exemple, avec des géométries de type LINESTRING

```
SELECT p.numero_parc, c.id_tuyau  
FROM conduites AS c, parcelles AS p  
WHERE (ST_Intersects(c.the_geom, p.the_geom)) ;
```

numero_parc character vary	id_tuyau character varying(40)
129	T 001
129	T 002
130	T 002

1. Liste des conduites traversant chaque parcelle



```
SELECT p.numero_parc, count(c.id_tuyau) as nb_tuyaoux  
FROM conduites AS c, parcelles AS p  
WHERE (ST_Intersects(c.the_geom, p.the_geom))  
GROUP BY p.numero_parc;
```

numero_parc character vary	nb_tuyaoux bigint
129	2
130	1

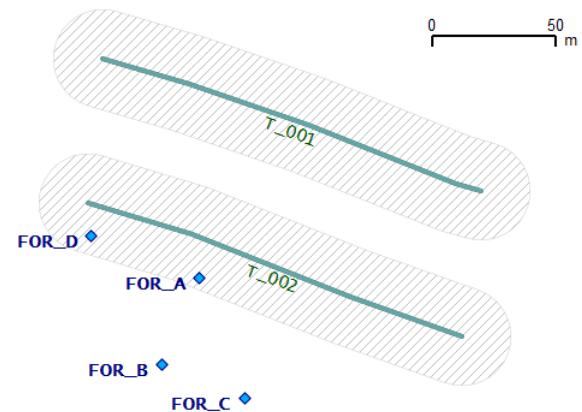
2. Nombre de conduites traversant chaque parcelle

# Autre exemple : la fonction ST\_DWithin (est compris dans un rayon de $d$ autour de $\text{geom}$ )

```
SELECT f.nom_forage, c.id_tuyau
FROM forages AS f, conduites AS c
WHERE (ST_DWithin(f.the_geom, c.the_geom, 20));
```

nom_forage character varying(80)	id_tuyau character varying(40)
FOR A	T 002
FOR D	T 002

Liste des forages situés  
dans un rayon de 20 m.  
autour des conduites



# Relations spatiales : les autres fonctions

## Liste des fonctions

`ST_Contains(geometry A, geometry B)` : retourne TRUE si aucun des points de B n'est à l'extérieur de A, et au moins un point de l'intérieur de B est à l'intérieur de A.

`ST_Crosses(geometry A, geometry B)` : retourne TRUE si la géométrie A a certains, mais pas la totalité, de ses points à l'intérieur de B.

`ST_Disjoint(geometry A , geometry B)` : retourne TRUE si les géométries ne s'intersectent pas - elles n'ont aucun point en commun.

`ST_Distance(geometry A, geometry B)` : retourne la distance cartésienne en 2 dimensions minimum entre deux géométries dans l'unité de la projection.

`ST_DWithin(geometry A, geometry B, radius)` : retourne TRUE si les géométries sont distante (radius) l'une de l'autre.

`ST_Equals(geometry A, geometry B)` : retourne TRUE si les géométries fournies représentent la même géométrie. L'ordre des entités n'est pas pris en compte.

`ST_Intersects(geometry A, geometry B)` : retourne TRUE si les géométries s'intersectent - (ont un espace en commun) et FALSE si elles n'en ont pas (elles sont disjointes).

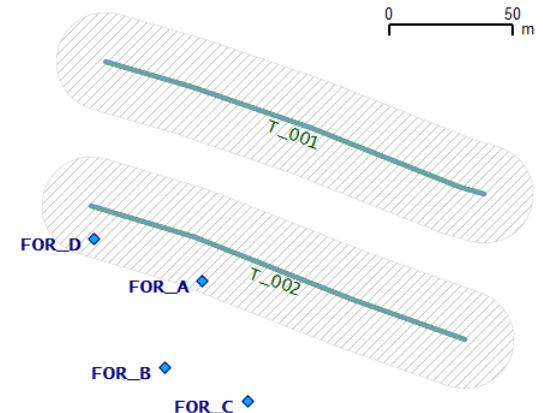
`ST_Overlaps(geometry A, geometry B)` : retourne TRUE si les géométries ont un espace en commun, sont de la même dimension, mais ne sont pas complètement contenues l'une dans l'autre.

`ST_Touches(geometry A, geometry B)` : retourne TRUE si les géométries ont au moins un point en commun, mais leur intérieurs ne s'intersectent pas.

`ST_Within(geometry A , geometry B)` : retourne TRUE si la géométrie A est complètement à l'intérieur de B

# Le problème du plus proche voisin

- Questions du type : pour chaque géométrie de A, quelle est la géométrie de B la plus proche ?
  - Quelle est la conduite la plus proche de chaque forage ?
  - Quel est le forage le plus proche de chaque autre forage ?
- Il faut mesurer la distance entre chaque paire de géométrie A et B ! Et retenir la plus courte distance



```
SELECT DISTINCT ON (f.nom_forage) f.nom_forage, c.id_tuyau,
ST_Distance(f.the_geom, c.the_geom)
FROM forages AS f, conduites as c
ORDER BY f.nom_forage, ST_Distance(f.the_geom, c.the_geom);
```

*DISTINCT ON (expression [...] ) keeps only the first row of each set of rows where the given expressions evaluate to equal.*

- Astuce optimisation : limiter la recherche à un certain rayon avec **ST\_DWithin**

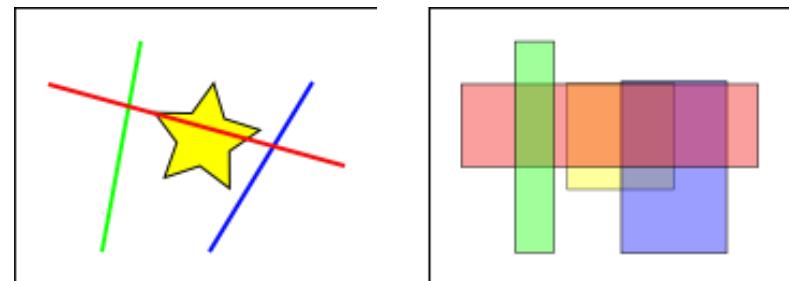
nom_forage character varying(80)	id_tuyau character varying(40)	st_distance double precision
FOR A	T 002	15.34141089526
FOR B	T 002	53.59241796147
FOR C	T 002	53.92560850901
FOR D	T 002	12.34385644098

[http://www.bostongis.com/PrinterFriendly.aspx?content\\_name=postgis\\_nearest\\_neighbor](http://www.bostongis.com/PrinterFriendly.aspx?content_name=postgis_nearest_neighbor)

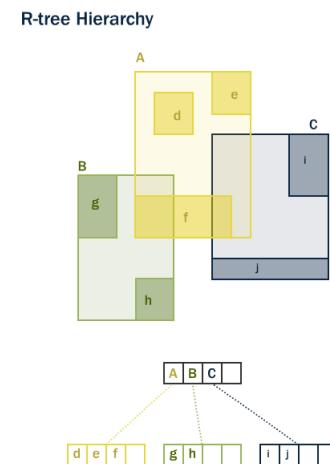
<http://blog.opengeo.org/2011/09/28/indexed-nearest-neighbour-search-in-postgis/>

# Utilisation des index spatiaux pour l'optimisation des requêtes spatiales

- Comment optimiser les requêtes spatiales et jointures spatiales ?
  - Exemple : quelles communes sont intersectées par la rivière R ?
- Les emprises géographiques (bounding boxes) permettent de faire une présélection « à moindre frais » avant les calculs plus complexes
  - De plus, les index spatiaux regroupent les emprises d'entités voisines sous forme de 'grappes' (R-Tree)



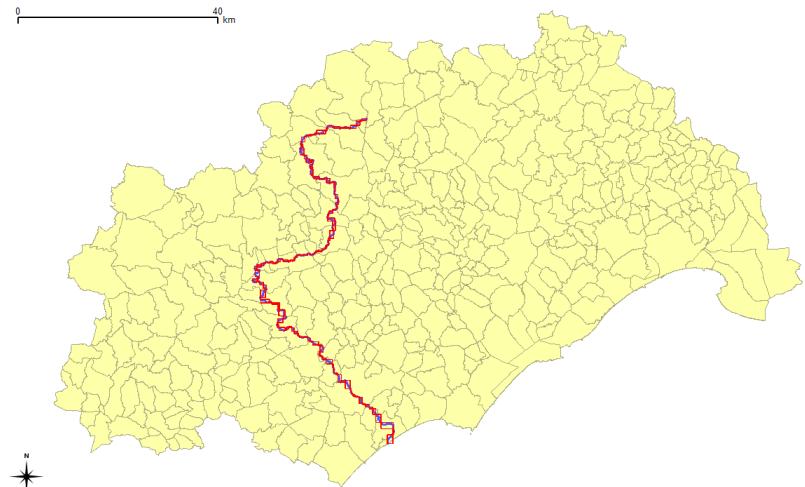
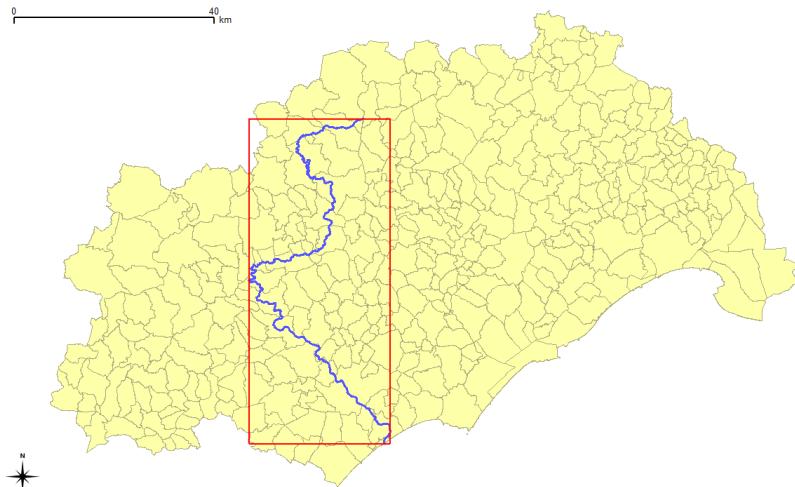
```
CREATE INDEX parcelles_the_geom_gist ON
public.parcelles USING gist(the_geom);
VACUUM ANALYZE parcelles;
```



# Modélisation des géométries et optimisation des requêtes

Nous cherchons la liste des communes traversées par le fleuve Orb.  
Dans quelle mesure l'index spatial accélèrera cette requête ?

- Ici, le fleuve Orb composé d'une seule entité très étendue (**1 ligne** dans la table)
- L'index spatial permettra t-il d'optimiser la requête efficacement ?
- Là, le fleuve Orb est composé de multiples tronçons courts (**203 lignes** dans la table)

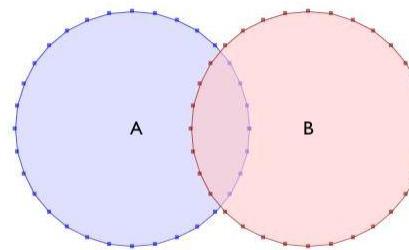


Formation PostGIS 1.5

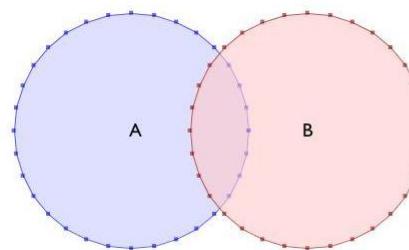
# **GEOTRAITEMENTS**

# Analyse spatiale et fonctions de construction géométrique

- 3 principales fonctions de construction géométrique
- **ST\_Intersection(geometry, geometry)** pour déterminer la surface commune (aussi pour croiser *lignes* et *surfaces*, *lignes* et *lignes*)
- **ST\_Union(geometry, geometry)** ou **ST\_Union(geometry[])** pour déterminer la surface totale ou la longueur totale (agréger des polygones, agréger des lignes)
- **ST\_Buffer(geometry, float)** pour déterminer la surface de  $d$  dans un rayon autour de  $g$



ST\_Intersection(A,B)



ST\_Union(A,B)

ST\_Buffer



Buffering a point



Buffering a multipoint



Buffering a linestring



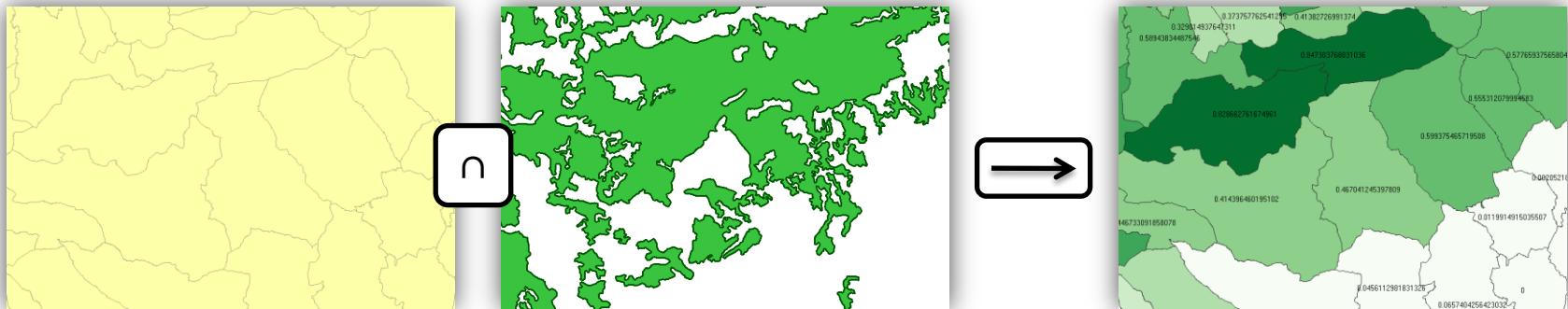
Buffering a polygon  
with one interior ring

[http://postgis.refractions.net/documentation/manual-1.5/reference.html#Geometry\\_Processing](http://postgis.refractions.net/documentation/manual-1.5/reference.html#Geometry_Processing)

[http://workshops.opengeo.org/postgis-intro/geometry\\_returning.html](http://workshops.opengeo.org/postgis-intro/geometry_returning.html)

# Analyse spatiale et fonctions de construction géométrique : exemples

- Les fonctions de construction géométrique **ST\_Intersection**, **ST\_Union**, **ST\_Buffer** permettent de croiser plusieurs tables pour analyser et surfaces et longueurs communes
- Comme « Outils d'analyse » dans ArcGIS et « Outils de géotraitements » dans QGIS...
- Exemple 1 : trouver la longueur de conduites traversant chaque parcelle
- Exemple 2 : trouver la surface occupée par les forêts de feuillus (*CORINE LAND COVER*) dans chaque commune (*BD TOPO*)
- Exemple 3 : déterminer la proportion de chaque type d'occupation du sol (*CORINE LAND COVER*) dans un rayon de 2 km autour de l'Orb (*BD Carthage*)



# Exemple 1

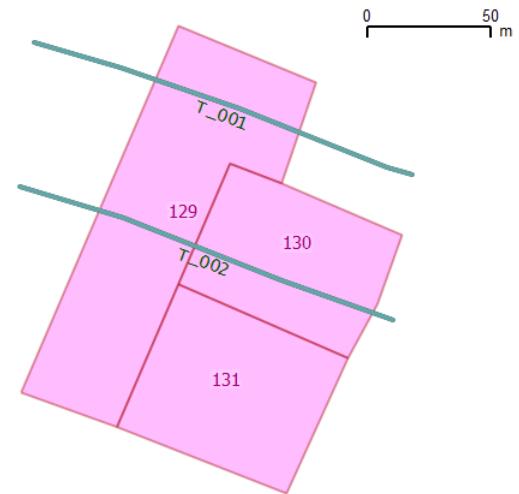
## Utilisation de la fonction ST\_Intersection

Nous cherchons la longueur totale de tuyaux traversant chaque parcelle

```
SELECT c.id_tuyau, p.numero_parc,  
       ST_Length(ST_Intersection(c.the_geom, p.the_geom)) AS longueur_m  
FROM conduites AS c, parcelles AS p  
WHERE (ST_Intersects(c.the_geom, p.the_geom));
```

id_tuyau character varying	numero_parc character varying	longueur_m double precision
T 001	129	62.2794102374
T 002	129	41.5041620190
T 002	130	77.3157949293

1. Intersection entre conduites et parcelles :  
calcul de la longueur



```
SELECT p.numero_parc,  
       sum(ST_Length(ST_Intersection(c.the_geom, p.the_geom))) AS total_long_m  
FROM conduites AS c, parcelles AS p  
WHERE (ST_Intersects(c.the_geom, p.the_geom))  
GROUP BY p.numero_parc;
```

numero_parc character varying	total_long_m double precision
129	103.783572256
130	77.3157949293

2. Pour chaque parcelle, calculer la somme de la longueur  
des parties de conduites qui traversent la parcelle

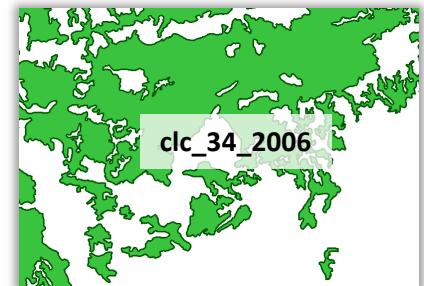
# Exemple 2 : intersection

## Occupation du sol / Limites administratives

Nous cherchons le ratio (surface ‘Forêts feuillus (CLC 311)’ / surface commune) pour chaque commune

1. Créer une table temporaire avec le total surface ‘feuillus’ par communes

« pour chaque commune, calcule la somme de la surface des polygones (commune  $\cap$  feuillus) »



```
-- table temp : surface clc 311 par communes
CREATE TEMP TABLE temp_clc_311_communne AS
SELECT b.gid, sum(ST_Area(ST_Intersection(c.the_geom, b.the_geom))) / 10000 AS surf_ha_311
FROM bdt_34_communes AS b, clc_34_2006 AS c
WHERE (c.code_06 = '311' AND ST_Intersects(c.the_geom, b.the_geom))
GROUP BY b.gid;
```

# Exemple 2 (suite et fin)

Nous disposons maintenant d'une table temporaire : surface 'Forêts feuillus (CLC 311)' par commune  
Calculons la proportion feuillus par commune dans la table bdt\_34\_communes

2. Jointure attributaire pour enregistrer surface feuillus (ha) dans une nouvelle colonne de la table bdt\_34\_communes

```
-- ajouter champ surf 311 dans bdt_34_communes
ALTER TABLE bdt_34_communes
ADD COLUMN surf_ha_311 double precision DEFAULT 0,
ADD COLUMN p_surf_311 double precision DEFAULT 0;
```

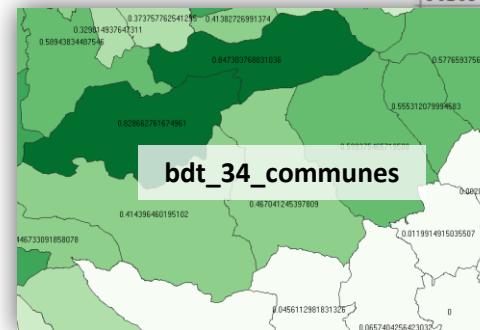
```
UPDATE bdt_34_communes AS b
SET surf_ha_311 = t.surf_ha_311
FROM temp_clc_311_commun AS t
WHERE (b.gid = t.gid);
```



3. Calcul du champ proportion feuillu = surface feuillus / surface commune

```
UPDATE bdt_34_communes
SET p_surf_311 = surf_ha_311 / (ST_Area(the_geom) / 10000));

-- voir resultat
SELECT code_insee, nom, p_surf_311
FROM bdt_34_communes
ORDER BY p_surf_311 DESC;
```

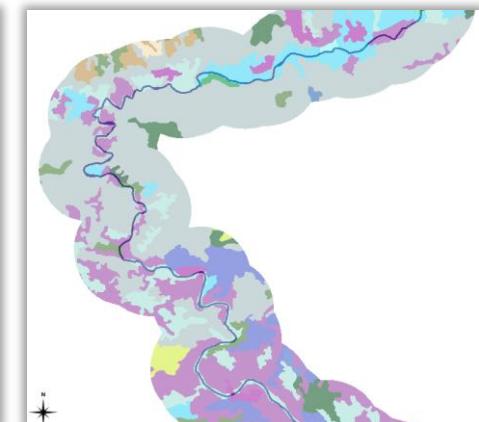
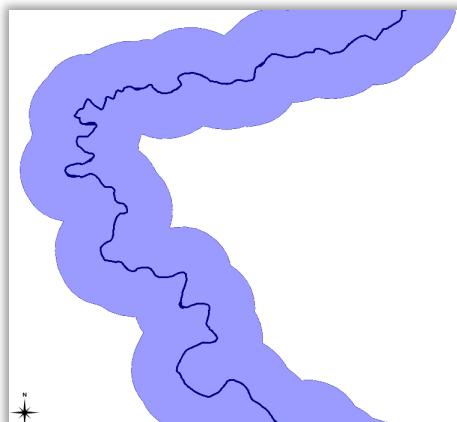
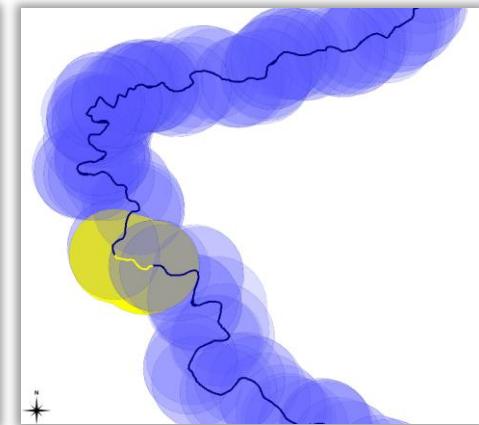
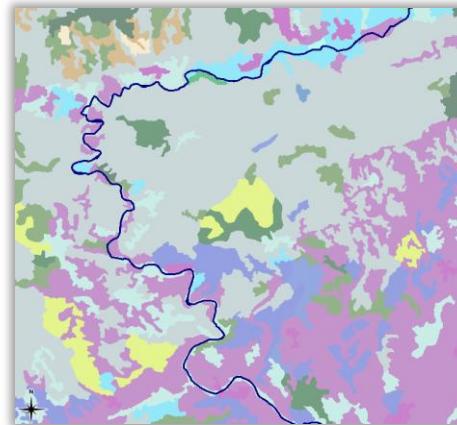


code_insee	nom	p_surf_311
character varying(5)	character varying(45)	double precision
34008	Les Aires	0.847303768031
34334	Vieuissan	0.828662761674
30280	Saint-Laurent-le-Minier	0.817668815355
34323	Valmascle	0.787623146873
34168	Montesquieu	0.747668298628
	Saint-Étienne-d'Albagnan	0.745319859580
	Saint-Étienne-Estréchoux	0.737819914647
	Saint-Julien-de-la-Nef	0.709865459136
	Rieussec	0.701402093830
	Olargues	0.692451076194
	Camplong	0.667988849578
	Ferrières-Poussarou	0.666128376667
	Pardailhan	0.665749920511
	Combes	0.650784330513
	résultat_Mines	0.648176496105

# Exemple 3 : utilisation de ST\_Buffer

Nous cherchons les différents types d'occupation du sol (surface ha et pourcentage surface) dans un rayon de 2 km autour de l'Orb

- Il faudra d'abord créer une zone tampon (dans une vue ou une table temporaire) autour des tronçons du fleuve Orb : **ST\_Buffer** et **ST\_Union**
- Ensuite, calculer l'intersection de cette zone tampon avec Corine Land Cover : **ST\_Intersection**



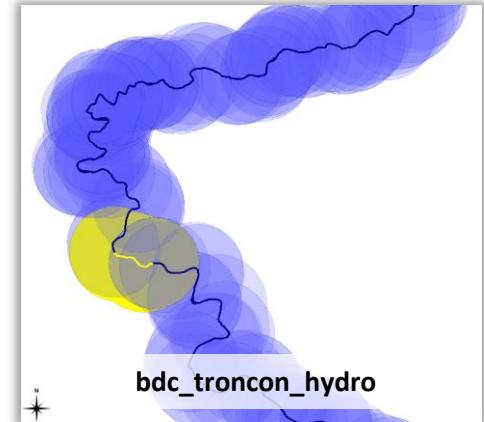
# Exemple 3 (suite)

## création et fusion des zones tampons

1. Nous créons les buffers non-fusionnés dans la même table que les tronçons hydro :  
ajout d'une colonne géométrique MULTIPOLYGON à la table bdc\_troncon\_hydro.

Remarquer l'utilisation de ST\_Multi pour convertir les POLYGON en MULTIPOLYGON

```
-- création buffer
SELECT AddGeometryColumn('public', 'bdc_troncon_hydro',
  'geom_buffer', 2154, 'MULTIPOLYGON', 2);
UPDATE bdc_troncon_hydro
SET geom_buffer = ST_Multi(ST_Buffer(the_geom, 2000))
WHERE toponymel = 'fleuve 1''orb';
```



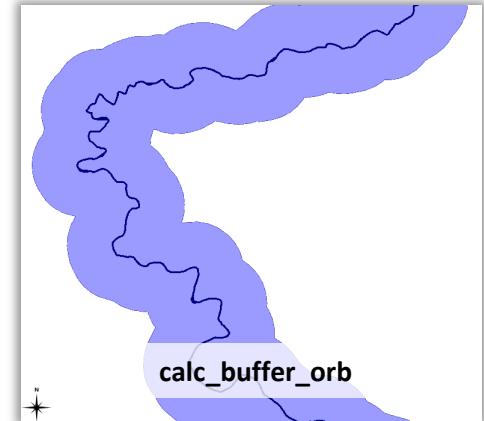
2. Fusion des multiples buffers dans un seul buffer : nous utilisons ST\_Accum pour créer un tableau de géométrie, pour ST\_Union pour fusionner ce tableau en une seule géométrie

```
-- union tous les buffers en un seul
CREATE TABLE calc_buffer_orb AS
SELECT 1::integer as gid,
  ST_Union(ST_Accum(geom_buffer)) as geom_buffer_orb
FROM bdc_troncon_hydro
WHERE toponymel = 'fleuve 1''orb';
```

<http://blog.cleverelephant.ca/2009/01/must-faster-unions-in-postgis-14.html>

3. L'utilisation de Populate\_Geometry\_Columns permet de référencer la table calc\_buffer\_orb dans la table geometry\_columns (pour l'afficher dans QGIS)

```
SELECT Populate_Geometry_Columns ('calc_buffer_orb' :: regclass);
```



# Exemple 3 (suite et fin)

## Intersection Zones tampons / Occupation du sol

Nous avons maintenant la zone tampon dans la table calc\_buffer\_orb.  
Reste à intersecer Corine Land Cover avec ce buffer et calculer les surfaces ...

### 4. Calculer la surface de chaque poste de CLC.

```
-- intersection buffer hydro / clc
SELECT code_06, sum(ST_Area(ST_Intersection(c.the_geom,
b.geom_buffer_orb))) / 10000 AS surf_ha_totale
FROM calc_buffer_orb AS b, clc_34_2006 AS c
WHERE ST_Intersects(c.the_geom, b.geom_buffer_orb)
GROUP BY code_06
ORDER BY surf_ha_totale DESC;
```

« pour chaque code CLC,  
calcule la somme de la surface  
des polygones (buffer  $\cap$  CLC) »

code_06 character varying(3)	surf_ha_totale double precision
311	13745.07904262
221	8790.994222460
242	7147.892400634
243	2737.915449121
112	2632.978636352
324	1884.286621435
312	1847.109980855
313	1507.235945501
323	1300.520812207
322	1026.895078247
231	871.4255695312
523	790.1735251953
211	660.2571822265

### 5. TODO : calculer le pourcentage occupé par chaque poste = surface du poste / surface totale

**FIN**